**Тема 3. Об'єктно-орієнтоване програмування**

**Лекція 12. Потоковий ввід/вивід при роботі з консоллю**

Навчальною метою розділу є ознайомлення читача з потоковим вводом/виводом даних до консолі. Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

– застосовувати оператори **cin** та **cout** для організації вводу та виводу;

– форматувати інформацію за допомогою маніпуляторів формату;

– використовувати вивчені підходи реалізації вводу/виводу у відповідності до поставленного завдання.

**12.1. Потоковий ввід/вивід**

Окрім вже розглянутого форматного вводу/виводу, що було успадковано від С, мова С++ дозволяє використовувати потоковий ввід/вивід на основі операторів **cin** та **cout**.

Потоком вводу/виводу при роботі з програмним забезпеченням виступає логічний пристрій, що застосовується як інтерфейс для впорядкованої передачі даних. Потоки вводу та виводу у С++ незалежні один від одного. Реалізація кожного з них виконується через свій оператор, згідно з чим іноді потік та оператор поєднують під одним поняттям: потік вводу **cin** та потік виводу cout. Робота з потоками здійснюється через операції вилучення ">>" та вставки "<<", при чому вилучення застосовується лише до потоку вводу, а вставка – до виводу. Таким чином, через програмний код зазначається, що дані, що вводить користувач вилучаються з потоку вводу консолі, а дані з програми вставляються у потік виводу на консоль. Використання потокового вводу/виводу при роботі зі змінними єпростішим у порівнянні з форматними, бо через потоки програма сама розуміє який тип даних вводиться або виводиться.

Формати потокових вводу та виводу наведено нижче:

**cin** >> <*змінна*> >> ...

**cout** << <*змінна*> <<...

Задавши потік одного разу, можна впродовж рядка скільки завгодно вилучати чи вставляти дані, чергуючи змінні з відповідними операціями.

Як і для форматного вводу/виводу у потоковому є можливість форматування тексту. Для цієї задачі використовуються маніпулятори, повний перелік яких наведено у табл. 12.1.

Таблиця 12.1 ***Маніпулятори***





Для того, щоб компілятор міг опрацьовувати код, який вміщує роботу з потоками, треба підключити наступні заголовні файли:

iostream.h – бібліотека, яка включає визначення операторів **cin** та **cout**

iomanip.h – бібліотека, що включає визначення маніпуляторів Причому підключення можна виконувати без зазначення розширення ".h", якщо планується задіяти простір імен std.

Розглянемо використання потокового вводу/виводу на прикладі вже знайомої за вивченням форматного вводу/виводу програмою, що запитує у користувача значення і виводить їх на консоль.

#**include** <iostream> //*підключення бібліотеки вводу/виводу*

#**include** <iomanip> //*підключення бібліотеки маніпуляторів*

**using** namespace std; //*застосування простору імен*

**void** main() //*оголошення та визначення головноїфункції*

{

**int** i; //*оголошення цілоїзмінної*

**float** f; //*оголошення змінноїз плаваючою точкою*

**char** c;

**cout** <<"Enter int value:" <<**flush**; //*вивід запиту до користувача*

**cin** >>i; //*отримання значення від користувача*

**cout** <<"int value is:" <<i <<**endl**; //*вивід значення на консоль*

**cout** <<"Enter float value:" <<**flush**; //*вивід запиту до користувача*

**cin** >>f; //*отримання значення від користувача*

//*вивід значення та наступного запиту до користувача*

**cout** <<"float value is:" <<f <<**endl**<<"Enter char value:" <<**flush**;

**cin** >>c; //*отримання значення від користувача*

**cout** <<"char value is:" <<c <<**endl**; //*вивід значення на консоль*

}

*Для роботи з потоковим вводом/виводом та маніпуляторами підключено бібліотеки* iostream *та* iomanip*. Необхідність задіяти простір імен* std *вказується за допомогою директиви* **using***. Програма під час виконання почергово запитує значення для цілої, з точкою, що плаває, та символьної змінних, і після їх вводу виводить отримані дані на консоль.*

У наведеному прикладі показано, що маніпулятори, наразі **endl**, можуть бути використані як у середині, так і наприкінці рядка потоку.

Якщо провести аналогію між двома способами вводу/виводу, що було розглянуто, то потоковий є більш зручнішим при роботі з даними, але при форматуванні він значно програє. Це пов'язано з тим, що при використанні маніпуляторів у потоці зміни діють постійно на відміну від форматних функцій, де форматування виконується тільки над поточним рядком. Окрім того, для простих дій потрібно застосовувати більше одного маніпулятору. Але форматування за допомогою маніпуляторів має переваги, бо воно є більш гнучким.

**Висновки**

У даному розділи було розглянуто наступні основні питання:

– вивід на консоль за допомогою потокового оператора cout;

– ввід з консолі за допомогою потокового оператора cin;

– маніпулятори форматування.

**Контрольні питання**

1) Що називають потоком вводу/виводу?

2) За допомогою якого оператора реалізується потоковий вивід даних на консоль?

3) Наведіть формат потокового виводу даних.

4) Для чого застосовують маніпулятори?

5) Які операції використовують для роботи з потоками вводу/виводу?

6) Який заголовний файл містить визначення для роботи потокових операторів вводу/виводу?

7) За допомогою якого оператора реалізовано потоковий ввід даних з консолі?

8) У чому полягає різниця у застосуванні маніпуляторів **endl** та **flush**?

9) Який заголовний файл містить визначення для роботи з маніпуляторами?

10) Порівняйте форматний та потоковий ввід/вивід.

**Лекція 13. Об'єктно-орієнтований підхід. класи та їх члени**

Навчальною метою розділу є ознайомлення читача з об'єктно- орієнтованим підходом при розробці програмного забезпечення та класами.

Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

– створювати класи;

– оголошувати та застосовувати об'єкти для роботи з членами класів;

– використовувати спеціальні методи класів;

– виконувати ініціалізацію полів класу за допомогою конструктора;

– реалізовувати інкапсуляцію;

– обмежувати доступ до членів класу за допомогою специфікаторів.

**13.1. Об'єктно-орієнтоване програмування**

В житті ми часто зустрічаємо поняття об'єкта. Так називають предмет, явище чи процес, що існує і має унікальні властивості, які його характеризують, та може бути задіяний тим чи іншим шляхом. Об'єкт може бути основою для проведення досліджень або слугувати метою для розробок.

З життя це поняття було запозичено і до програмування. Але програмний об'єкт не є існуючим насправді. Так прийнято називати сукупність даних та функцій для роботи з ними, що згідно із вказаною задачею характеризують об'єкт, що є реальним. Завдячуючи такій реалізації на базі програмного забезпечення, можна створювати моделі, що достатньо чітко повторюють процеси та явища. Окрім того, на відміну від процедурних мов програмування, об'єктно-орієнтований підхід надає можливість поєднання підпрограм та даних за призначенням у окремі модулі.

**13.1.1. Клас**

Основою при написанні об'єктно-орієнтованих програм є клас. Він реалізує функцію інкапсуляції, яка надає програмісту можливість використовувати структурні блоки коду через вбудований інтерфейс, не вдаючись у подробиці змісту їх побудови. Інкапсуляція є одним з трьох базових понять об'єктно-орієнтованого програмування.

Класом прийнято називати сукупність даних, поєднаних за загальним змістом, та функцій, що дозволяють застосовувати ці дані. Вони дуже схожі на структури. Ті й інші є складеними типами даних, можуть включати змінні стандартних типів даних та реалізовуватися через екземпляри та змінні-покажчики. Екземпляри класів при об'єктно-орієнтованому програмуванні часто називають об'єктами, що дозволяє відрізняти їх як окремі змінні при описі коду та зазначає їх призначення.

Створення класів виконується за допомогою службового слова **class** за наступним форматом:

**сlass** <*ім'я-класу>*

{

< *оголошення функцій та змінних >*

*<специфікатор\_доступу 1>:*

*< оголошення функцій та змінних>*

*<специфікатор\_доступу 2>:*

*< оголошення функцій та змінних>*

*<специфікатор\_доступу N>:*

*< оголошення функцій та змінних>*

}<*список об'єктів>*;

Список об'єктів як і для структур не є обов'язковим і застосовується лише тоді, коли є необхідність у створенні екземпляра глобальної видимості. Ім'я класу зазвичай обирається, зважаючи на відповідне його призначення. Кожне слово починається з великої літери і попереду дописується "С", що відповідає англійському class. Такий підхід дозволяє використовувати класи як програмні модулі, що можуть бути застосовані багатьма розробниками, як компоненти програм для вирішення великої кількості різноманітних завдань. Для цього оголошення класу виділяють у файл з розширенням ".h", а визначення всіх його методів у файл ".срр" робить клас окремим структурним елементом (див. Багатофайлові проекти).

**13.1.2. Доступ до членів класу**

Функції та змінні, що включені до класу, є його членами, причому функції прийнято називати методами, а змінні, як і для структур, – полями.

Часто говорять, що дані інкапсульовані у клас.

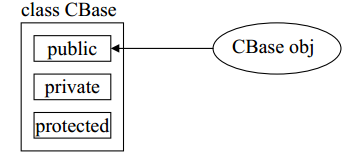
Доступ до полів та методів класу з основної програми може бути трьох видів: відкритий, закритий та захищений. Вид доступу визначається за допомогою спеціальних службових слів, що називаються специфікаторами доступу:

**public** – повний доступ (відкритий);

**private** – доступ лише з методів класу (закритий);

**protected** – можливість доступу з методів членів та з похідного класу (захищений).

Таким чином, робота з членами класу здійснюється за схемою, що наведено на рис. 13.1.



***Рис. 13.1. Схематичне відображення доступу до членів класу***

Якщо при оголошенні членів класу специфікатор доступу не було вказано, то за умовчуванням застосовується закритий доступ.

Визначення методів класу можна виконувати як у самому класі, так і за його межами. При цьому до імені функції, що визначається, зліва через операцію "::" додається ім'я її класу.

<*тип даних* > <*ім'я класу*>::<*ім'я методу класу*>(<*аргументи функції*>)

{

<*тіло функції*>

}

Операцію "::" називають глобальним дозволом. Вона застосовується для того, щоб вказати належність функцій чи даних до відповідного класу.

Розглянемо приклад класу, що поєднує дані та функції роботи з відрізком на площині.

**class** CSegment //*оголошення класу для робот із відрізком*

{

**public**: //*відкритий доступ*

**int** x0; //*абсциса першоїточки відрізка*

**int** y0; //*ордината першоїточки відрізка*

**int** x1; //*абсциса другоїточки відрізка*

**int** y1; //*ордината другоїточки відрізка*

//*оголошення та визначення функції отримання координат точки центра*

//*відрізка*

**void** GetCenterPoint(**int**& x, **int**& y)

{

x=(x1+x0)/2; //*розрахунок абсциси точки центра*

y=(y1+y0)/2; //*розрахунок ординати точки центра*

}

//*оголошення функціївизначення довжини відрізка по осі абсцис*

**int** GetXLength();

//*оголошення функціївизначення довжини відрізка по осі ординат*

**int** GetYLength();

};

//*визначення функціївизначення довжини відрізка по осі абсцис*

**int** CSegment::GetXLength()

{

**return** (x0<x1)?(x1-x0):(x0-x1);

}

//*визначення функціївизначення довжини відрізка по осі ординат*

**int** CSegment::GetYLength()

{

**return** (y0<y1)?(y1-y0):(y0-y1);

}

*В оголошеному класі членами є поля* x0*,* x1*,* y0*,* y1 *для зберігання значень координат точок початку та кінця відрізка та методи* GetCenterPoint() *– отримання координати центра відрізка,* GetXLength() *– отримання довжини відрізка вздовж осі* Х*,* GetYLength() *– отримання довжини відрізка вздовж осі* Y*. Оголошення методу* GetCenterPoint() *виконано у класі, а* GetXLength() *та* GetYLength() *за його межами.*

Застосування методів та полів класу у програмі відповідає роботі зі структурами. Спочатку потрібно оголосити об'єкт, а потім скористатися операцією доступу до членів – ".". Наприклад, використання розглянутого вище класу може полягати у наступному коді:

CSegment s; //*оголошення об'єкта класу*

//*визначеня координати першоїточки*  
s.x0=10;

s.y0=25;

//*визначеня координати другоїточки*

s.x1=20;

s.y1=49;

**int** x,y; //*оголошення змінних для отримання координати центра*

//*виклик методу для отримання координати центра відрізка*

s.GetCenterPoint(x,y);

//*вивід на консоль отриманоїкоординати*

**cout**<<"Center point: ("<<x<<","<<y<<")"<<**endl**;

//*отримання і вивід на консоль довжини відрізка вздовж осі абсцис*

**cout**<<"X Length: "<<s.GetXLength()<<**endl**;

//*отримання і вивід на консоль довжини відрізка вздовж осі ординат*

**cout**<<"Y Length: "<<s.GetYLength()<<**endl**;

**13.1.3. Спеціальні методи класу**

Два методи класу є особливими, бо не можуть існувати як окремі функції.

Це конструктор та деструктор. Вони не мають типу даних і іменуються, як і сам клас. Різниця у їх назвах полягає лише у символі "~", що додається до назви деструктора.

Конструктор – це метод класу, який викликається автоматично при створенні об'єкта. Він використовується для визначення початкових значень полів, виділення пам'яті та захоплення ресурсів.

Деструктор теж викликається автоматично, але при знищенні об'єкта класу. У деструкторі виконуються дії, необхідні для коректного завершення роботи з даними та методами класу, – вивільнення пам'яті чи ресурсів.

Обидва методи існують у класах навіть, коли їх не було зазначено, бо становлять невід'ємну частину об'єктів. Але без явного оголошення та визначення конструктор та деструктор не можна використовувати для вирішення завдань програміста.

Особливістю використання конструктора є також список ініціалізації, що дописується після закриваючої круглої дужки списку аргументів та застосовується для визначення полів.

Автоматичний виклик спеціальних методів означає, що програміст не має можливості викликати їх де завгодно у програмному коді. Конструктор викликається в точці оголошення об'єкта і через нього може отримувати значення для своїх аргументів. На відміну від нього деструктор зовсім не має аргументів, бо можливість їх визначення відсутня. З автоматичним викликом також пов'язана відсутність у спеціальних методів значень, що повертаються. Їх не можливо було б отримати з програми. Крім того, конструктор і деструктор повинні при оголошенні бути включені до членів з відкритим доступом. Спроба застосувати інший специфікатор доступу призведе до помилки.

Як і для звичайних функцій, для методів, за виключенням деструктора, працює властивість перевантаження, що значно розширює можливості застосування класів. Перевантаження деструктора не може бути виконано за причиною відсутності в нього списку аргументів.

Розглянемо приклад програми, яка працює з класом, що дозволяє програмісту застосовувати пов'язаний з об'єктом динамічний масив.

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

#**include** <iomanip> //*підключення бібліотеки маніпуляторів*

**using namespace** std; //*застосування простору імен std*

**class** CDynArr //*оголошення класу роботи з динамічним масивом*

{

**int** size; //*оголошення поля для зберігання розміру масиву*

**public**: //*відкриті члени класу*

**int**\* pArr; //*змінна-покажчик для створення масиву*

//*оголошення конструктора класу без аргументів*

CDynArr() : size(10) //*список ініціалізації*

{

pArr=**new int**[size]; //*створення динамічного масиву*

//*вивід статичного рядка на консоль*

**cout**<<"Arrey with size 10 was created"<<**endl**;

}

//*оголошення конструктора класу з аргументом для визначення розміру масиву*

CDynArr(**int** sz) : size(sz) //*список ініціалізації*

{

pArr=**new int**[size]; //*створення динамічного масиву*

//*вивід статичного рядка на консоль*

**cout**<<"Arrey with size "<<sz<<" was created"<<**endl**;

}

//*оголошення методу запису значень у масив*

**bool** AddVal(**int**, **int**);

//*оголошення методу читання значень елементів масиву*

**bool** GetVal(**int**, **int**&);

//*оголошення методу отримання максимального значення масиву*

**int** GetMaxVal();

//*оголошення методу отримання мінімального значення масиву*

**int** GetMinVal();

//*оголошення методу отримання середнього арифметичного значення масиву*  
**float** GetMidVal();

//*оголошення деструктора класу*~CDynArr()

{

**delete** [] pArr; //*вивільнення пам'яті*

//*вивід статичного рядка на консоль*

**cout**<<"Arrey with size "<<size<<" was deleted"<<**endl**;

}

};

//*визначення методу запису значень у масив*

**bool** CDynArr::AddVal(**int** pos, **int** val)

{

**if** (pos > (size-1) || pos < 0) //*якщо вказаний елемент відсутній*

**return** 0; //*повернути 0*

\*(pArr+pos)=val; //*записати значення до елемента*

**return** 1; //*повернути 0*

}

//*визначення методу читання значень елементів масиву*

**bool** CDynArr::GetVal(**int** pos, **int**& val)

{

**if**(pos > (size-1) || pos < 0) //*якщо вказаний елемент відсутній*

**return** 0; //*повернути 0*

val=\*(pArr+pos); //*читати значення елемента*

**return** 1; //*повернути 0*

}

//*визначення методу отримання максимального значення масиву*

**int** CDynArr::GetMaxVal()

{

//*оголошення змінноїдля пошуку максимального значення*

**int** max=\*pArr; //*визначення змінноїпершим значенням*

**for**(int i=1; i < size; i++) //*цикл перебору всіх елементів з 2*

**if**(max < \*(pArr+i)) //*якщо значення більше за max*

max=\*(pArr+i); //*визначити max*

**return** max; //*повернути отримане значення*

}

//*визначення методу отримання мінімального значення масиву*

**int** CDynArr::GetMinVal()

{

//*оголошення змінноїдля пошуку мінімального значення*

**int** min=\*pArr; //*визначення змінноїпершим значенням*

**for**(int i=1; i < size; i++)//*цикл перебору всіх елементів з 2*

**if**(min > \*(pArr+i)) //*якщо значення менше за min*

min=\*(pArr+i); //*визначити min*

**return** min; //*повернути отримане значення*

}

//*визначення методу отримання середнього арифметичного значення масиву*

**float** CDynArr::GetMidVal()

{

//*оголошення змінноїдля розрахунку суми та їївизначення*

**float** sum=0;

**for**(int i=0; i < size; i++) //*цикл перебору всіх елементів*

sum+=\*(pArr+i); //*додавання поточного значення*

**return** sum/size; //*повернути середнє арифметичне*

}

//*оголошення та визначення головноїфункціїпрограми*

**void** main()

{

**int** v; //*оголошення змінноїдля отримання значень користувача*

**cout**<<"Auto size:"<<**endl**; //*вивід на консоль статичного рядка*

//*оголошення змінної-покажчика на клас*

CDynArr \*pda; //*оголошення змінної-покажчика на клас*

pda=**new** CDynArr; //*створення динамічного об'єкту*

**int** i=0; //*оголошення та визначення змінноїлічильника*

//*цикл заповнення масиву об'єкта*

**Do**

{

//*вивід на консоль рядка запиту*

**cout**<<"Enter "<<i<<" value: "<<**flush**;

//*отримання значення від користувача*

**cin**>>v;

//*визначення елемента масиву та перевірка умови циклу*

} **while**(pda->AddVal(i++, v));

//*вивід на консоль знайденого максимального значення*

**cout**<<"Max value is: "<< pda->GetMaxVal()<<**endl**;

//*вивід на консоль отриманого мінімального значення*

**cout**<<"Min value is: "<< pda->GetMinVal()<<**endl**;

//*вивід на консоль отриманого середнього арифметичного значення*

**cout**<<"Mid value is: "<< pda->GetMidVal()<<**endl**;

**delete** pda; //*вивільнення пам'яті*

**cout**<<"User size:"<<**endl**; //*вивід на консоль статичного рядка*

//*вивід на консоль запиту до користувача*

**cout**<<"Enter the arrey size (1...): "<<**flush**;

**cin**>>v; //*отримання кількості елементів масиву*

**if**(v > 0) //*якщо значення більше за 0*

{

i=0; //*визначення змінноїлічильника*

//*оголошення об'єкта та визначення аргументу конструктора*

CDynArr da(v);

//*цикл заповнення масиву об'єкта*

**do**

{

//*вивід на консоль рядка запиту*

**cout**<<"Enter "<<i<<" value: "<<**flush**;

//*отримання значення від користувача*

**cin**>>v;

//*визначення елемента масиву та перевірка умови циклу*

}

**while**(da.AddVal(i++, v));

//*вивід на консоль знайденого максимального значення*

**cout**<<"Max value is: "<< da.GetMaxVal()<<**endl**;

//*вивід на консоль отриманого мінімального значення*

**cout**<<"Min value is: "<< da.GetMinVal()<<**endl**;

//*вивід на консоль отриманого середнього арифметичного значення*

**cout**<<"Mid value is: "<< da.GetMidVal()<<**endl**;

}

**else** //*якщо значення менше чи дорівнює 0*

//*вивід на консоль статичного рядка*

**cout**<<"Wrong size!"<<**endl**;

}

*Клас включає поля* size *та* pArr *для створення динамічного масиву і методи* CDynArr() *– конструктори класу,* ~CDynArr() *– деструктор класу,* AddVal() *– запис значення у масив,* GetVal() *– читання значення з масиву,* GetMaxVal() *– отримання максимального значення серед елементів,* GetMinVal() *– отримання мінімального значення серед елементів,* GetMidVal() *– отримання середнього арифметичного для значень масиву. У залежності від того, було чи ні при оголошенні об'єкта вказано значення аргументу, виконується виклик першої чи другої реалізації конструктора. Список ініціалізації використовується задля визначення поля size. У тілах конструкторів виконується створення динамічного масиву. Деструктор вивільнює пам'ять, що була задіяна під масив.*

*У головній функції робота з класом демонструється двома шляхами – через створення динамічного та статичного об'єктів. При створенні динамічного об'єкта виконується виклик конструктора без аргументів, а при створенні статичного – з аргументом.*

Призначення класів є дуже широким, бо їх застосування не тільки виконує задачу функцій структуризації програми, але ще й робить код модульним, що застосовується при розробці великих проєктів, до яких задіяно більш одного програміста. Крім того, інкапсуляція та можливість створення окремих закритих бібліотек дозволяють використовувати класи як комерційний продукт для продажу.

**Висновки**

У даному розділи було розглянуто наступні основні питання:

– об'єктно-орієнтований підхід в програмуванні;

– класи та їх члени;

– спеціальні методи класів.

**Контрольні питання**

1) Дайте визначення поняттю клас.

2) Що називають інкапсуляцією?

3) У чому полягає призначення об'єктів класу?

4) Які методи класу є спеціальними?

5) Який метод зазвичай використовується для визначення даних класу за умовчуванням при створенні об'єкта?

6) Який метод призначено для вивільнення ресурсів при знищенні об'єкта?

7) Для чого використовується список ініціалізації?

8) Яке призначення має перевантаження конструктора?

9) Чому спеціальні методи не мають значень, що повертаються?

10) Чому деструктор не може бути перевантажено?

11) Для яких задач може знадобитися перевантаження конструктора?

12) Що означає автоматичний виклик спеціальних методів?

13) Як можна визначити аргументи конструктора?

14) У чому полягає схожість між класом та структурою?

15) Як виконується визначення методу класу за його межами?

**Лекція 14. Спадкування**

Навчальною метою розділу є ознайомлення читача зі спадкуванням властивостей одного класу іншим.

Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

– створювати зв'язок спадкування між класами;

– використовувати специфікатори доступу при підключенні базового класу до похідного;

– розробляти ієрархічну структуру класів;

– передавати значення до аргументів базового класу через об'єкт похідного;

– перевантажувати методи базового класу у похідному;

– виконувати множинне спадкування;

– запобігати невизначеності при спадкуванні.

**14.1. Спадкування при об'єктно-орієнтованому програмуванні**

Спадкування є другим з трьох базових понять об'єктно-орієнтованого програмування. Так прийнято називати процес, при якому частина властивостей одного класу переходить до іншого. Перший клас іменують базовим, а другий – похідним. Сутність процесу полягає у тому, що при створенні об'єкта класу спадкоємця в ньому, окрім своїх членів, також створюється копія відкритих та захищених членів базового класу. Таким чином, похідний клас переймає від базового частину членів, а також має свої унікальні дані та методи. Цей підхід дозволяє поширювати можливості класів шляхом побудови ієрархічної структури, у якій кожен спадкоємець включає лише ті здібності, що необхідні для доповнення простого базового об'єкта до більш складного.

**14.1.1. Створення похідних класів**

Похідним називають клас, що успадковує від іншого відкриті та захищені члени. Формат оголошення похідного класу наведено нижче:

**class** <*ім'я класу*> : <*специфікатор доступу*> <*ім'я базового класу*>

{

< *оголошення функцій та змінних >*

*<специфікатор доступу 1>:*

< *оголошення функцій та змінних >*

*<специфікатор доступу 2>*

< *оголошення функцій та змінних >*

*.*

*.*

*.*

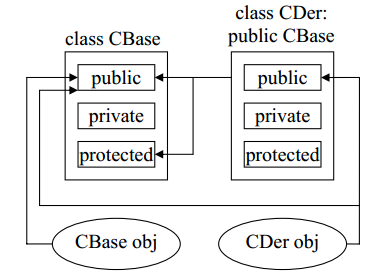
*<специфікатор доступу N>:*

< *оголошення функцій та змінних >*

}<*список об'єктів>*;

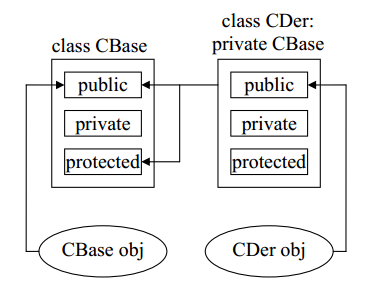
Різниця оголошень звичайного та похідного класів полягає лише у списку спадкування, що зазначається після імені через оператор ":". При підключенні базових класів використовуються вже відомі специфікатори доступу **private**, **public**, **protected**, що визначають вид доступу до копії членів.

Якщо базовий клас підключено через специфікатор **public**, то для методів похідного класу відкрито всі поля та методи базового класу, що мають відкритий та захищений види доступу. Через об'єкт класу можна застосовувати всі відкриті члени як похідного, так і базових класів. Робота з таким класом відповідає схемі, наведеній на рис. 14.1.



***Рис. 14.1. Схематичне відображення доступу до членів базового та похідного класів при відкритому спадкуванні***

Якщо ж базовий клас було підключено через специфікатор **private**, то методам похідного класу, як і при **public,** доступні всі відкриті та захищені члени, а для об'єкта лише відкриті поля та методи похідного класу. Робота з таким класом відповідає схемі, наведеній на рис. 14.2.



***Рис. 14.2. Схематичне відображення доступу до членів базового та похідного класів при закритому спадкуванні***

Підключення базового класу через специфікатор доступу **protected** відповідає схемі з закритим підключенням. Різниця з'являється лише при черговому спадкуванні.

Таким чином, специфікатори доступу при підключенні базового класу до похідного слугують для розширення інкапсуляції членів.

Розглянемо клас, що є похідним від розробленого раніше CSegment, і використовується для роботи з чотирикутником на площі:

//*оголошення похідного класу та підключення до нього базового через private*

**class** CQuaderangle : **private** CSegment

{

**public**: //*члени з відкритим доступом*

**int** Xm[4]; //*оголошення масиву для збереження абсцис точок*

**int** Ym[4]; //*оголошення масиву для збереження ординат точок*

//*оголошення конструктора з аргументами*

CQuaderangle (**int** Ax, **int** Ay, **int** Bx, **int** By,

**int** Cx, **int** Cy,**int** Dx, **int** Dy)

{

//*виклик методу для перевизначення координат вершин чотирикутника*

ReInitPoints(Ax,Ay,Bx,By,Cx,Cy,Dx,Dy);

}

//*оголошення конструктора без аргументів*

CQuaderangle ()

{

//*виклик методу для перевизначення координат вершин чотирикутника*

ReInitPoints(-10,20,10,20,100,-20,-100,-20);

}

//*оголошення методу для перевизначення координат вершин чотирикутника*

**void** ReInitPoints(**int**, **int**, **int**, **int**, **int**, **int**, **int**, **int**);

//*оголошення методу, що перевіряє чи є фігура правильною*

**bool** IsEquilateral();

//*оголошення методу, що повертає координату точки центра фігури*

**void** GetFigureCenter(**int**& x, **int**& y);

};

//*визначення методу для перевизначення координат вершин чотирикутника*

**void** CQuaderangle::ReInitPoints(**int** Ax, **int** Ay, **int** Bx, **int** By,

**int** Cx, **int** Cy, **int** Dx, **int** Dy)

{

//*визначення координати першоївершини чотирикутника*  
Xm[0]=Ax;

Ym[0]=Ay;

//*визначення координати другоївершини чотирикутника*

Xm[1]=Bx;

Ym[1]=By;

//*визначення координати третьоївершини чотирикутника*

Xm[2]=Cx;

Ym[2]=Cy;

//*визначення координати четвертоївершини чотирикутника*

Xm[3]=Dx;

Ym[3]=Dy;

}

//*визначення методу, що первірє чи є фігура правильною*

**bool** CQuaderangle::IsEquilateral()

{

//*перевірка на те, чи має фігура чотири кути*

**if**((Xm[0]!=Xm[1] || Ym[0]!=Ym[1]) &&

(Xm[0]!=Xm[2] || Ym[0]!=Ym[2]) &&

(Xm[0]!=Xm[3] || Ym[0]!=Ym[3]) &&

(Xm[1]!=Xm[2] || Ym[1]!=Ym[2]) &&

(Xm[1]!=Xm[3] || Ym[1]!=Ym[3]) &&

(Xm[2]!=Xm[3] || Ym[2]!=Ym[3]))

{

//*оголошення змінних для отримання координат*

**int** fv0,fv1,sv0,sv1;

//*визначення координат точок протилежних вершин*

x0=Xm[0];

y0=Ym[0];

x1=Xm[2];

y1=Ym[2];

//*визначення координати центра першоїдіагоналі*

GetCenterPoint(fv0, fv1);

//*визначення координат точок протилежних вершин*

x0=Xm[0];

y0=Ym[0];

x1=Xm[2];

y1=Ym[2];

//*визначення координати центра другоїдіагоналі*

GetCenterPoint(sv0, sv1);

//*якщо точки центрів діагоналей збігаються*

**if**(fv0==sv0 && fv1==sv1)

{

//*визначення координати точки перетину діагоналей*

x0=fv0;

y0=fv1;

//*визначення координати точки першоївершини*

x1=Xm[0];

y1=Ym[0];

//*порахування суми довжин відрізка вздовж Ох та Оу*

sv0=GetXLength()+GetYLength();

//*визначення координати точки другоївершини*

x1=Xm[1];

y1=Ym[1];

//*порахування суми довжин відрізка вздовж Ох та Оу*

sv1=GetXLength()+GetYLength();

**if**(sv0==sv1) //*якщо отримані значення збігаються*

//*повернути підтвердження правильності фігури*

**return** 1;

**return** 0; //*повернути спростування правильності фігури*

}

**return** 0; //*повернути спростування правильності фігури*

}

**return** 0; //*повернути спростування правильності фігури*

}

//*визначення методу, що повертає координату точки центра фігури*

**void** CQuaderangle::GetFigureCenter(**int**& x, **int**& y)

{

x=y=0; //*визначення початкових значень для змінних*

//*цикл перебору всіх координат*

**for**(**int** i=0; i<4; i++)

{

x+=Xm[i]; //*додати до змінноїзначення абсциси точки*

y+=Ym[i]; //*додати до змінноїзначення ординати точки*

}

x/=4; //*порахувати абсцису точки центра*

y/=4; //*порахувати ординату точки центра*

}

*Клас включає наступні члени: два поля масивів по чотири елементи, для збереження точок вершин чотирикутника, та методи* CQuaderangle() *– конструктори класу для визначення початкових значень полів,* ReInitPoints() *– для перевизначення значень полів,* IsEquilateral() *– для первірки на те, що чотирикутник є правильним,* GetFigureCenter() *– для отримання координат точки центра фігури. Сутність успадкування для наведеного класу полягає в тому, що у тілі функції* IsEquilateral() *застосовані методи базового класу.*

*Підключення* CSegment *до* CQuaderangle *відбувається за допомогою специфікатора доступу* **private***, що робить члени базового класу закритими для застосування через об'єкти похідного класу.*

Важливо розуміти, що на основі процесу спадкування встановлюється лише однобічний зв'язок. Базовий клас не має жодного доступу до похідного.

Робота з похідними класами за допомогою змінних-покажчиків нічим не відрізнється від звичайних класів.

**14.1.2. Виклик конструктора та деструктора базового класу**

Спеціальні методи у похідних класах грають ту саму роль, що і у звичайних. Особливість стосується лише копії базового класу. Конструктор та деструктор завжди мають відкритий доступ, що приводить до їх обов'язкового копіювання у похідний клас. Виклик їх залишається автоматичним, але стає пов'язаним зі створенням об'єкта класу спадкоємця. Таким чином, для похідного класу існують два конструктори та два деструктори. Конструктор копії викликається перед конструктором самого класу, а деструктор копії після деструктора класу.

Щоб зрозуміти черговість автоматичних викликів спеціальних методів при спадкуванні, розглянемо приклад програми з використанням базового та похідного класів, код конструкторів та деструкторів котрих виводить на консоль відповідний до події статичний рядок:

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

#**include** <iomanip> //*підключення бібліотеки застосування маніпуляторів*

**using namespace** std; //*застосування простору імен std*

**class** CBase //*оголошення базового класу*

{

**public**: //*відкриті члени класу*

CBase() //*оголошення та визначення конструктора*

{

//*вивід статичного рядка на консоль*

**cout**<<"Call of CBase Constructor"<<**endl**;

}

~CBase() //*оголошення та визначення деструктора*

{

//*вивід статичного рядка на консоль*

**cout**<<"Call of CBase Destructor"<<**endl**;

}

};

//*оголошення похідного класу з відкритим підключенням базового*

**class** CDer : **public** CBase

{

**public**: //*відкриті члени класу*

CDer() //*оголошення та визначення конструктора*

{

//*вивід статичного рядка на консоль*

**cout**<<"Call of CDer Constructor"<<**endl**;

}

~CDer() //*оголошення та визначення деструктора*

{

//*вивід статичного рядка на консоль*

**cout**<<"Call of CDer Destructor"<<**endl**;

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

CDer d; //*оголошення об'єкту похідного класу*

}

*Для роботи з потоковим вводом/виводом підключені бібліотеки iostream та iomanip. Код головної функції включає лише оголошення об'єкта похідного класу, що приводить до автоматичних викликів конструкторів. При досягненні закриваючої фігурної дужки об'єкт знищується, що відповідно викликає деструктори. Результат програми наступний:*

Call of CBase Constructor

Call of CDer Constructor

Call of CDer Destructor

Call of CBase Destructor

Особливість викликів пов'язано з тим, що при реалізації любого методу похідного класу копія базового повинна вже існувати.

**14.1.3. Визначення аргументів конструктора базового класу**

Як вже було зазначено, конструктори достатньо часто мають аргументи, які визначаються при оголошенні об'єктів. Але визначити аргументи конструктора базового класу через об'єкт похідного неможливо. Це пов'язано з тим, що його оголошення, як і для звичайних класів, може слугувати лише задля визначення аргументів свого власного конструктора.

Визначення параметрів конструктору базового класу може бути здійснено лише за одним шляхом – через список ініціалізації конструктора похідного класу. Це єдине місце у програмі, де можна виконати виклик автоматичного методу конструктора.

Розглянемо приклад програми, в якій виконується визначення аргументу конструктора базового класу, що в свою чергу ініціює закрите поле.

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

#**include** <iomanip> //*підключення бібліотеки застосування маніпуляторів*

**using namespace** std; //*застосування простору імен std*

**class** CBase //*оголошення базового класу*

{

**int** val; //*оголошення закритого поля*

**public**: //*відкриті члени класу*

//*оголошення та визначення конструктора, визначення поля*CBase(**int** a) : val(a) //*визначення поля через список ініціалізації*

{

}

//*оголошення та визначення методу для виводу значення поля на консоль*

**void** ShowValue()

{

//*вивід на консоль значення поля*

**cout**<<"val="<<val<<**endl**;

}

};

//*оголошення похідного класу з відкритим підключенням базового*

**class** CDer : **public** CBase

{

**public**: //*відкриті члени класу*

//*оголошення та визначення конструктора*

CDer(int b) : CBase(b) //*виклик конструктора CBase*

{

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

**int** a; //*оголошення змінноїдля отримання значення від користувача*

//*вивід на консоль запиту до користувача*

**cout**<<"Enter value for private val:"<<**flush**;

**cin**>>a; //*отримання введеного значення*

CDer d(a); //*оголошення об'єкта та визначення аргументу*

//*виклик копіїметоду базового класу виводу значення поля*

d.ShowValue();

}

*Для роботи з потоковим вводом/виводом підключені бібліотеки iostream та iomanip. Базовий клас включає закрите поле для зберігання значення цілої змінної та два методи – конструктор і функцію виводу значення поля на консоль. Визначення поля виконується через список ініціалізації конструктора за допомогою значення, отриманого через аргумент. Похідний клас включає лише конструктор з аргументом, що використовується для передачі значення до конструктора базового класу, що викликається у списку ініціалізації. У головній функції виконується запит до користувача на ввід значення. Після вводу воно використовується для визначення аргументу об'єкта похідного класу. Значення закритого поля val виводиться на консоль за допомогою виклику копіїметоду базового класу.*

**14.2. Перевантаження методів базового класу у похідному**

Перевантаження методів при спадкуванні дещо відрізняється від звичайного перевантаження. При спадкуванні є допустимим створити метод, який не тільки має однакове зі скопійованим ім'я, але й таку саму кількість та типи аргументів.

Розглянемо приклад програми, яка працює з похідним класом, що має перевантажений метод для розрахунку суми чисел від 1 до N з кроком в 1.

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

#**include** <iomanip> //*підключення бібліотеки застосування маніпуляторів*

**using namespace** std; //*застосування простору імен std*

**class** CBase //*оголошення базового класу*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення методу для розрахунку суми значень від 1 до val*

**int** GetSumFromTo(**int** val)

{

//*вивід на консоль статичного рядка, що визначає належність методу*

**cout**<<"CBase function – "<<**flush**;

**int** sum=0; //*оголошення змінноїта їїпочаткове визначення*

**do** //*цикл розрахунку суми*

{

sum+=val--; //*додавання значення до суми*

} **while**(val>0); //*умова виконання циклу*

**return** sum; //*повернення отриманого значення*

}

};

**class** CDer : **public** CBase //*оголошення похідного класу*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення методу для розрахунку суми значень від 1 до val*

**int** GetSumFromTo(**int** val)

{

**if**(val>1) //*якщо значення більше за 1*

//*повернення значення рекурсивного виклику методу похідного класу*

**return** val+GetSumFromTo(val-1);

//*вивід на консоль статичного рядка, що визначає належність методу*

**cout**<<"CDer function – "<<**flush**;

**return** 1; //*повернення значення 1*

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

**int** v; //*оголошення змінноїдля збереження значень користувача*

CDer d; //*оголошення об'єкта класу*

//*вивід на консоль запиту до користувача*

**cout**<<"Enter first value for calculating sum: "<<**flush**;

//*отримання введеного значення*

**cin**>>v;

//*вивід на консоль результату виклику методу похідного класу*

**cout**<<"Rezult is: "<<d.GetSumFromTo(v)<<**endl**;

//*вивід на консоль запиту до користувача*

**cout**<<"Enter second value for calculating sum: "<<**flush**;

//*отримання введеного значення*

**cin**>>v;

//*вивід на консоль результату виклику методу базового класу*

**cout**<<"Rezult is: "<<d.CBase::GetSumFromTo(v)<<**endl**;

}

*Код програми починається з підключення бібліотек для роботи з потоковим вводом/виводом. Потім виконується оголошення базового та похідного класів, кожен з яких має метод* GetSumFromTo() *з одним цілим аргументом. У базовому класі сума розраховується за допомогою циклу з постумовою, а у похідному через рекурсивний виклик. У головній функції користувачеві виводяться два запроси на ввід значень, для кожного з яких сума розраховується через різні методи. При вводі значень* 5 *та* 7 *результат виконання програми буде наступним:*

Enter first value for calculating sum: 5

CDer function – Rezult is: 15

Enter first value for calculating sum: 7

CBase function – Rezult is: 28

Якщо через об'єкт похідного класу викликається перевантажений метод, що має повністю однакові оголошення, то компілятором автоматично обирається функція, яка не є скопійованою. Виклик іншої версії функції може бути виконано лише через операцію глобального дозволу "::", що зазначить клас, метод якого повинно бути викликано.

Перевантаження при спадкуванні використовується не лише для застосування різних підпрограм, але й для розширення можливостей методів, що вже існують. У наведеному вище прикладі для методу похідного класу застосовано властивість рекурсивного виклику. Цей код свідчить, що навіть з функцій у похідному класі перевантажений метод базового класу звичайним чином викликати неможливо, лише через операцію глобального дозволу. Розглянемо, програму, що відповідає дещо зміненому попередньому прикладу. Вона не має двох реалізацій розрахунку суми. Перевантаження методу у похідному класі виконується задля запобігання можливим помилкам користувача, пов'язаним з вводом від'ємних значень чи нуля.

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

#**include** <iomanip> //*підключення бібліотеки застосування маніпуляторів*

**using namespace** std; //*застосування простору імен std*

**class** CBase //*оголошення базового класу*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення методу для розрахунку суми значень від 1 до val*

**int** GetSumFromTo(**int** val)

{

**int** sum=0; //*оголошення змінноїта їїпочаткове визначення*

**do** //*цикл розрахунку суми*

{

sum+=val--; //*додавання значення до суми*

} **while**(val>0); //*умова виконання циклу*

**return** sum; //*повернення отриманого значення*

}

};

**class** CDer : **public** CBase //*оголошення похідного класу*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення методу для розрахунку суми значень від 1 до val*

**int** GetSumFromTo(**int** val)

{

**if**(val>=1) //*якщо значення більше за одиницю*

//*повернути результат виконання методу базового класу*

**return** CBase::GetSumFromTo(val);

//*вивід на консоль статичного рядка*

**cout**<<"Wrong value! – "<<**flush**;

**return** 0; //*повернення значення 0*

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

**int** v; //*оголошення змінноїдля збереження значень користувача* CDer d; //*оголошення об'єкта класу*

//*вивід на консоль запиту до користувача*

**cout**<<"Enter first value for calculating sum: "<<**flush**;

**cin**>>v; //*отримання введеного значення*

//*вивід на консоль результату виклику методу похідного класу*

**cout**<<"Rezult is: "<<d.CBase::GetSumFromTo(v)<<**endl**;

//*вивід на консоль запиту до користувача*

**cout**<<"Enter second value for calculating sum: "<<**flush**;

//*отримання введеного значення*

**cin**>>v;

//*вивід на консоль результату виклику методу базового класу*

**cout**<<"Rezult is: "<<d.GetSumFromTo(v)<<**endl**;

}

*Різниця у наведених кодах полягає у тому, що перевантажений метод не виконує окрему підпрограму, в його тілі перевіряється коректність отриманого значення і при відповідності викликається метод базового класу.*

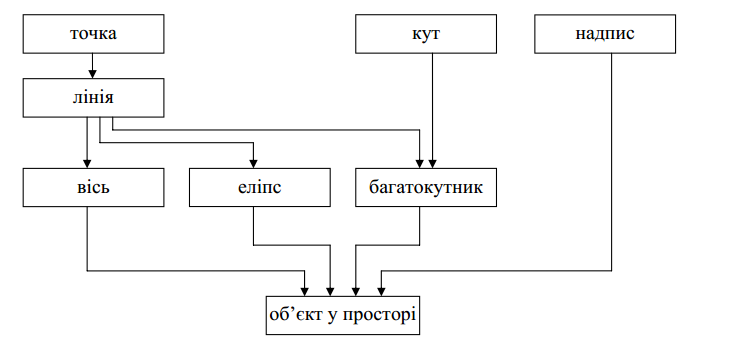
Перевантаження при спадкуванні використовується достатньо часто. Цей механізм дозволяє виконувати як дороботку вже існуючих методов, так і підвищує гнучкість ієрархічної структури класів.

**14.3. Множинне спадкування і сутність невизначеності при ньому**

Згідно з правилами спадкування, похідний клас може включати копії заразом декількох базових. Такий підхід називають множинним спадкуванням.

Це значно поширює можливості ієрархічного спадкування, бо один складний об'єкт може складатися з необхідної кількості простих. Крім того, прості класи є легкими в реалізації, тому для програміста при розробці складного проекту достатнім є лише виконати у правильній послідовності зв'язки між членами об'єктів.

Наприклад, за множинним наслідуванням може бути створено ієрархію класів для роботи з геометричними фігурами у просторі за схемою на рис. 14.3.



***Рис. 14.3. Ієрархія класів роботи з графічними об'єктами на площині***

Множинне спадкування виконується за форматом оголошення похідного класу. Базові класи підключають за допомогою застосування специфікаторів доступу через оператор "кома":

**class** <*ім'я класу*> : <*специфікатор доступу 1*> <*ім'я 1-го базового класу*>, <*специфікатор доступу 2*> <*ім'я 2-го базового класу*>. . .

Так, клас CDer, що є похідним від CA, CB та CC може бути оголошений за наступним прикладом:

**class** CDer : **public** CA, **private** CB, **public** CC

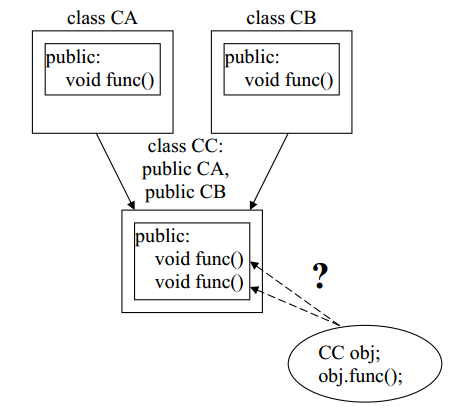
{};

Перевагою множинного спадкування є те, що воно значно розширює об'єктно-орієнтований підхід при написанні програмного забезпечення.

Недоліком можна вважати деякі додаткові витрати на планування, щоб запобігти невизначеності.

Невизначеністю при множинному спадкуванні називають ситуацію, коли компілятор не може без додаткового визначення зрозуміти, яку з копій перевантажених методів необхідно підставити у місце виклику.

Частіше за все невизначеність виникає у випадку, коли похідний клас спадкує від двох базових методи з однаковими іменами. Таким чином, у ньому існують заразом дві копії відповідної функції і, при її виклику, компілятору є незрозумілим, яку з підпрограм необхідно використати (рис. 14.4).



***Рис. 14.4. Схематичне відображення невизначеності при множинному спадкуванні***

Одним з шляхів вирішення невизначеності є вже розглянута операція глобального визначення – "::". Якщо при всіх викликах перевантажених методів буде використано ім'я відповідного класу, то перед компілятором не буде стояти необхідність вибору. Наприклад, для наведеної вище схеми:

CC obj; //*оголошення об'єкта похідного класу*

obj.CA::func(); //*виклик методу, що є копією з CA*

obj.CB::func(); //*виклик методу, що є копією з CB*

Ще один шлях вирішення невизначеностей полягає у застосуванні віртуалізації. Його буде розглянуто у розділі "Поліморфізм".

**Висновки**

У даному розділи було розглянуто наступні основні питання:

– спадкування класів;

– обмеження доступу при спадкуванні;

– переваження методів при спадкуванні;

– множинне спадкування;

– запобігання невизначеності при множинному спадкуванні.

**Контрольні питання**

1) Дайте визначення поняттю спадкування.

2) Назвіть та охарактеризуйте класи, що пов'язані через спадкування?

3) Як виконується підключення базового класу?

4) У чому різниця між спадкуванням через специфікатори відкритого та закритого доступу?

5) Як виконується виклик методів похідного класу через змінну-

покажчик?

6) В чому полягає особливість автоматичного виклику конструктора та деструктора базового класу при оголошенні об'єкта похідного класу?

7) Які члени базового класу і при якій умові доступні через об'єкт похідного класу?

8) Як виконується визначення аргументів конструктора базового класу через об'єкт похідного класу?

9) В чому полягає особливість списків ініціалізації стосовно до конструктора базового класу?

10) Дайте визначення поняттю перевантаження методів при спадкуванні.

11) Поясніть, чому при виклиці методу, що було визначено у базовому класі та перевантажено при спадкуванні, через об'єкт похідного класу реалізується лише функція похідного класу?

12) Як через об'єкт похідного класу викликати перевантажений метод базового класу?

13) У чому полягає призначення перевантаження методів при спадкуванні?

14) Дайте визначення множинному спадкуванню.

15) Що надає програмісту використання множинного спадкування?

16) Яку ситуацію називають невизначеністю при множинному спадкуванні?

17) Яка ситуація, що призводить до невизначеності при множинному спадкуванні, є найбільш поширеною?

**Лекція 15. Віртуальні, дружні та статичні функції**

Навчальною метою розділу є ознайомлення читача з поліморфізмом, дружніми функціями і класами, та статичними членами.

Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

– реалізовувати поліморфізм на базі віртуальних функцій;

– застосовувати чисті віртуальні функції для створення ієрархічної структури класів;

– виконувати виклик деструктора базового при роботі з динамічним об'єктом похідного класу;

– запобігати невизначеності при спадкуванні за допомогою віртуальних функцій та класів;

– застосовувати дружні функції та класи для роботи з закритими та захищеними членами класів;

– використовувати статичні члени класів для взаємодії між об'єктами.

**15.1. Поліморфізм**

Поліморфізм означає мінливість чи здібність до перетворювань. Для об'єктно-орієнтованого програмування поліморфізм – це властивість виклику однойменних методів похідних від одного базового, класів через змінну- покажчик на клас, що спадкується. Наприклад, припустимо при наявності ієрархічної побудови класів для відображення графічних фігур класи CTriangle, CSquare та CCircle є похідними від базового класу CFigure. В кожному з них визначено метод Draw(), що дозволяє зображувати фігуру на формі. Зручність застосування поліморфізму полягає у тому, що, створивши масив fig[3] змінних-покажчиків на клас CFigure і доповнивши його значеннями адрес об'єктів похідних класів, можна виконати відображення всіх трьох фігур за допомогою нижче наведеного лістинга:

**for**(**int** i=0; i<3; i++) //*цикл перебору індексів масиву*

f[i]->Draw(); //*виклик методу об'єкта*

Таким чином, опрацювання однойменних методів будь-якої кількості похідних від одного базового класів можна реалізувати, застосувавши лише один цикл, в якому будуть перебиратися адреси існуючих об'єктів.

Але використання властивості поліморфізму в об'єктно-орієнтованому програмуванні потребує, щоб базовий клас теж мав однойменний метод з похідними. Окрім того, метод базового класу повинен бути оголошений віртуальним. Лише в цьому випадку компілятор мови С++ зрозуміє, що є потреба перейти від раннього до пізнього зв'язування, яке і відкриє можливість застосування властивості поліморфізму.

**15.1.1. Віртуальні функції**

Слово віртуальний (virtual – англ.) означає: той, що є видимим, але не існуючим насправді. Віртуальною функція стає у тому випадку, коли при її оголошенні та визначенні було використано службове слово **virtual**. Формат оголошення віртуальної функції зазначено нижче:

**virtual** <*тип даних* > <*ім'я функції* > (<*список параметрів*>);

Розглянемо сутність використання віртуальних функцій на наступному прикладі: створимо базовий клас та два похідних від нього; визначимо в цих класах однойменні відкриті методи, що виконують різні дії; у функції main() реалізуємо виклик всіх трьох методів через єдину змінну-покажчик.

#**include** <iostream> //*підключення бібліотеки вводу/виводу*

**using** namespace std; //*застосування простору імен*

**class** CBase //*оголоення базового класу*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення методу виводу значення*

**void** show\_num()

{

**cout**<<"0"<<**endl**; //*вивід константного рядка*

}

};

**class** CDer1 : **public** CBase //*оголошення класу спадкоємця*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення методу виводу значення*

**void** show\_num()

{

**cout**<<"1"<<**endl**; //*вивід константного рядка*

}

};

**class** CDer2 : **public** CBase //*оголошення класу спадкоємця*

{

**public**: //*відкриті члени классу*

//*оголошення та визначення методу виводу значення*

**void** show\_num()

{

**cout**<<"2"<<**endl**; //*вивід константного рядка*

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

CBase \*pB,base; //*оголошення змінної-покажчика та об'єкта*

CDer1 der1; //*оголошення об'єкта*

CDer2 der2; //*оголошення об'єкта*

pB=&base; //*визначення змінної-покажчика адресою об'єкта*

pB->show\_num(); //*виклик методу*

pB=&der1; //*визначення змінної-покажчика адресою об'єкта*

pB->show\_num(); //*виклик методу*

pB=&der2; //*визначення змінної-покажчика адресою об'єкта*

pB->show\_num(); //*виклик методу*

}

*Класи* CDer1 *та* CDer2 *є похідними від* CBase*. В кожному з трьох класів створено метод* show\_num()*, що є перезавантаженим. Завдячуючи тому, що методи виводять на консоль унікальні числа, можна відстежити який з них було реалізовано. У головній функції послідовно до змінної-покажчика на* CBase *записуються адреси об'єктів всіх трьох класів та виконується виклик методу* show\_num()*.*

Треба зазначити, що таке написання програмного коду не є помилкою. Раніше було сказано, що неприпустимо визначати змінну-покажчик на один тип даних адресою змінної іншого типу даних, але об'єкти, що відповідають базовому та похідним від нього класам, компілятором вважаються змінними одного типу даних. Ця особливість об'єктно-орієнтованого програмування і дає можливість програмістам застосовувати властивість поліморфізму.

При виконанні програмного коду прикладу на консоль буде виведено наступний стовпець з цифр:

0

0

0

Це зовсім не відповідає очікуваному результатові. Компілятор не зміг визначити різниці між адресами об'єктів і кожного разу було виконано метод базового класу.

Внесемо до нашої програми невеличку зміну – нехай метод show\_num() базового класу стане віртуальним. Його визначення буде мати наступний вигляд:

//*оголошення та визначення віртуального методу виводу значення* **virtual void** show\_num()

{

**cout**<<"0"<<**endl**; //*вивід константного рядка*

}

Нове виконання програмного коду виведе на консоль наступний  
стовпець:

0

2

**15.1.2. Раннє та пізнє зв'язування**

Таким чином, якщо метод базового класу, що перезавантажено у похідних класах, буде оголошено як віртуальний, то компілятор починає розуміти різницю між адресами об'єктів. Хоча насправді це лише видимість.

Через змінну-покажчик, яку визначено адресою, компілятор не може дізнатися про тип об'єкта. Підхід, що застосовується у цьому випадку, називають пізнім зв'язуванням. Протилежне до нього раннє зв'язування є класичним при компіляції програм, в яких відсутні віртуальні методи чи класи. При ньому компілятор виконує зв'язування змінної-покажчика з методами та полями того класу, що зазначено при його оголошенні. Цей підхід було обрано при першому виконанні наведеного прикладу. На відміну від раннього, при пізньому зв'язуванні компілятор відкладає встановлення зв'язків до запуску програми, під час якого через існуючу явним чином адресу об'єкта застосовується відповідна функція. Це сталося під час другого виконання прикладу. Раннє та пізнє зв'язування також часто називають статичним та динамічним.

**15.1.3. Чисті віртуальні функції та абстрактні класи**

З наведеного вище прикладу можна зробити висновок, що при відтворенні поліморфізму у програмах базовий клас необов'язково повинен мати реалізацію для віртуальних методів. Зручно створити базовий клас з безліччю порожніх віртуальних методів для пізнього зв'язування, а всі особливості їх реалізації, пов'язані з призначенням, перекласти на перезавантажені у похідних класах однойменні методи. Таким чином, базовий клас з окремого елемента формування при застосуванні властивості поліморфізму перетворюється на шаблонну основу для організації ієрархічної структури похідних класів.

Щоб перейти до зазначеного застосування базового классу, використовуються чисті віртуальні функції. Віртуальна функція стає чистою віртуальною в тому випадку, коли при її оголошенні було застосовано наступний формат:

**virtual** <*тип даних* > <*ім'я функції*>(<*список параметрів*>)=0;

"=0" не пов'язано з присвоюванням значення нуля, це лише форма запису, що визначає віртуальну функцію як чисту віртуальну.

Визначати чисті віртуальні функції не треба, бо вони у програмі не можуть бути реалізовані. Спроба їх виклику призведе до помилки при компіляції. Окрім того, оголошення лише однієї чистої віртуальної функції має вплив на весь клас цілком. Всі методи такого класу автоматично стають чистими віртуальними, а він сам перетворюється на абстрактний. Для таких класів об'єктів існувати не може, лише змінні-покажчики. Основне призначення абстрактних класів полягає у формуванні на їх основі ієрархічних структур похідних класів для реалізації властивості поліморфізму.

**15.2. Віртуальний деструктор**

Якщо при написанні програм програміст застосовує властивість поліморфізму, коли працює з динамічно створеними об'єктами похідних класів і хоче, щоб при вивільненні пам'яті було викликано деструктор похідного класу, то деструктор базового класу обов'язково повинен бути оголошений як віртуальний.

Розглянемо приклад:

#**include** <iostream> //*підключення бібліотеки вводу/виводу*

**using** namespace std; //*застосування простору імен*

**class** CBase //*оголошення базового класу*

{

**public**: //*відкриті члени класу*

~CBase() //*оголошення та визначення деструктора*

{

**cout**<<"Base"<<**endl**; //*вивід константного рядка*

}

};

**class** CDer : **public** CBase

{

**public**: //*відкриті члени класу*

~CDer() //*оголошення та визначення деструктора*

{

**cout**<<"Derived"<<**endl**; //*вивід константного рядка*

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

CBase \*pB=**new** CDer; //*динамічне створення об'єкта*

**delete** pB; //*вивільненні пам'яті*

}

*В програмі створено базовий клас* CBase *та похідний від нього клас* CDer*. В кожному з класів визначено деструктор. При видаленні об'єкта відповідний деструктор виводить на консоль назву класу, до якого належить. У головній функції виконується динамічне створення об'єкта класу* CDer *та запис його адреси у змінну-покажчик на клас* CBase*. Наступним рядком об'єкт видаляється.*

Результатом виконання наведеної програми є:

Base

Це означає, що деструктор похідного класу було проігноровано. Такий підхід може призвести до значних помилок роботи програми. Змінимо простий деструктор базового класу на віртуальний:

**virtual** ~CBase() //*оголошення та визначення віртуального деструктора*

{

**cout**<<"Base"<<**endl**; //*вивід константного рядка*

}

Запуск нової версії програми приведе до появи на консолі рядків:  
Derived Base

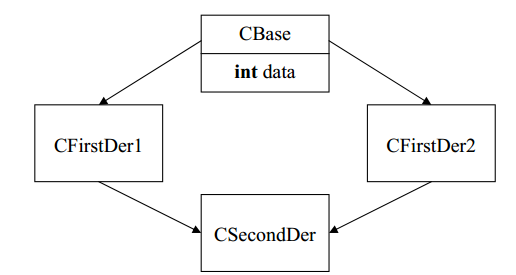
Цей результат повністю відповідає поставленій задачі.

Зазначимо, що необхідність у віртуалізації деструкторів зникає, коли об'єкти створено статичним шляхом.

**15.3. Віртуальні базові класи**

Ще одним призначенням віртуалізації, але вже не пов'язаним з поліморфізмом, є усунення невизначеності при множинному спадкуванні. Для цієї задачі використовуються віртуальні базові класи.

Таким чином, наведену на рис. 15.1 схему множинного спадкування, можливо реалізувати через віртуальні класи без невизначеності.



***Рис. 15.1. Схема множинного спадкування***

**class** CBase //*оголошення базового класу*

{

CBase

**int** data

CSecondDer

CFirstDer1 CFirstDer2

**public**: //*відкриті члени класу*

**int** data; //*оголошення цілоїзмінної*

};

**class** CFirstDer1 : **virtual public** CBase //*оголошення класу спадкоємця*

{

};

**class** CFirstDer2 : **virtual public** CBase //*оголошення класу спадкоємця*

{

};

//*оголошення класу спадкоємця*

**class** CSecondDer : **public** CFirstDer1, **public** CFirstDer2

{

**public**: //*відкриті члени класу*

**void** func(**int** i) //*оголошення та визначення методу*

{

data=i; //*визначення поля класу*

}

};

В цьому фрагменті програмного коду невизначеність відсутня, бо при підключенні базового класу до обох спадкоємців було зазначено, що він є віртуальним. Такий підхід не впливає на базовий клас, а лише говорить компілятору, що для всіх похідних класів буде існувати лише єдина копія класу, що було успадковано.

1. **5. Дружні функції та дружні класи**

Застосування інкапсуляції в об'єктно-орієнтованому програмуванні призводить до того, що глобальні функції, що не є методами класів, не спроможні мати доступ до їх полів. Іноді таке нерушиме правило може призвести до деяких незручностей при програмуванні, бо виникає необхідність у застосуванні закритих чи захищених полів класу у глобальній функції. В цьому випадку на допомогу приходить властивість створення дружніх до вказаного класів і функцій. Такі класи і функції можуть виконувати операції з інкапсульованими даними та методами без явного формування залежності від вказаного класу.

Зазначення дружності виконується за допомогою службового слова friend.

Для створення вказівки компілятору про те, що функція чи клас для поточного класу буде виступати у якості дружньої, треба в його тілі скористатися одним з наступних форматів:

**friend** <*тип даних* > <*ім'я функції* >(<*список параметрів* >);

**friend class** <*ім'я класу* >;

Розглянемо программу, у якій застосовано дружність функції ffunc() та класу CFriend до класу CTest з закритим полем data та методом inc().

#**include** <iostream> //*підключення бібліотеки вводу/виводу*

**using** namespace std; //*застосування простору імен*

**class** CTest //*оголошення класу*

{

**private**: //*закриті члени класу*

**int** data; //*оголошення поля збереження даних*

//*оголошення та визначення методу інкрементування поля*

**void** inc()

{

data++; //*інкрементування значення поля*

}

**public**: //*відкриті члени класу*

//*оголошення та визначення конструктора*

CTest() : data(5) //*визначення поля за умовчуванням*

{

}

**friend void** ffunc(CTest); //*оголошення дружньоїфункції*

**friend class** CFriend; //*оголошення дружнього класу*

};

//*визначення класу, що є дружнім*

**class** CFriend

{

**public**: //*відкриті члени класу*

**void** fmetod(CTest t) //*оголошення та визначення методу*

{

//*вивід значення поля на консоль*

**cout**<<"data is:"<<t.data<<**endl**;

t.inc(); //*інкрементування значення поля*

//*вивід значення поля на консоль*

**cout**<<"data+=1 is: "<<t.data<<**endl**;

}

};

**void** ffunc(CTest t) //*визначення функції, що є дружньою до класу*

{

**cout**<<"data is:"<<t.data<<**endl**; //*вивід значення поля на консоль*

t.inc(); //*інкрементування значення поля*

**cout**<<"data+=1 is: "<<t.data<<**endl**; //*вивід значення поля на консоль*

}

**void** main() //*оголошення та визначення головноїфункції*

{

CTest a; //*оголошення об'єкта класу*

ffunc(a); //*виклик дружньоїфункції*

CFriend f; //*оголошення об'єкта дружнього класу*

f.fmetod(a); //*виклик методу дружнього класу*

}

Клас CTest, окрім поля dataта метода inc(), має конструктор, у якому виконується ініціалізація data значенням 5, та рядки, що повідомляють компілятор про необхідність відкрити функції ffunc() та класу CFriend повний доступ як дружнім. Ці рядки не служать у якості оголошення відповідної функції та класу. Оголошення та визначення виконуються нижче. Функція ffunc() та єдиний метод класу CFriend мають однакову реалізацію: вивід на консоль двох рядків, поточного значення змінної data та її значення після інкрементування. У головній функції створюється об'єкт а класу CTest, а потім викликаються ffunc() та fmetod().

Запуск програми приводить до появи на консолі наступного результату роботи:

data is: 5 data+=1 is: 6 data is: 5 data+=1 is: 6

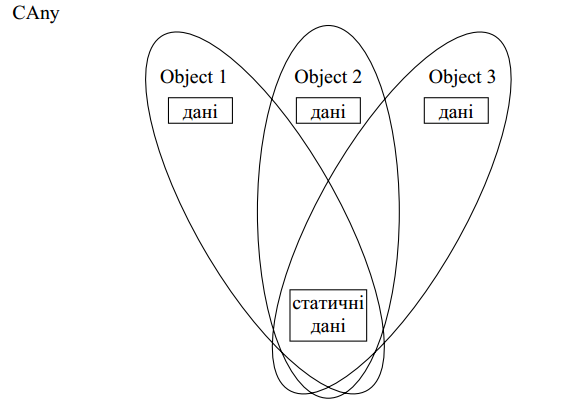
Таким чином, використання властивості дружності призводить до ігнорування інкапсуляції функціями та класами, що зазначені як дружні. Але цей підхід треба застосовувати обережно, бо зловживання дружністю може порушити стійкість структури об'єктно-орієнтованої програми.

**15.5. Статичні поля та функції**

При роботі з об'єктами класу іноді виникає необхідність в організації між ними зв'язків, що базуються на використанні спільних змінних. Цю функцію можуть виконувати глобальні змінні, але такий підхід не завжди є зручним. По- перше є недоліки застосування глобальних змінних, а по-друге порушується незалежність об'єктів класів від програми в цілому.

Правильне вирішення такої задачі полягає у використанні статичних членів класу. Їх застосування дозволяє організовувати зв'язок між окремими об'єктами та виконувати одночасну ініціалізацію, що є важливим при необхідності зміни параметрів налаштування об'єктів.

Особливістю статичних членів класу є те, що для всіх об'єктів існує лише єдина їх версія. Це наглядно демонструє схема на рис. 15.2.



***Рис. 15.2. Статичні дані відносно об'єктів класу***

Незалежно від кількості об'єктів статичні члени класу завжди будуть подані в одному екземплярі. У програмі для оголошення статичних полів та методів використовуються нижче наведені формати:

**static** <*тип даних* ><*ім'я поля* >;

**static** <*тип даних* ><*ім'я методу* >(<*список параметрів* >);

Розглянемо приклад, у якому реалізовано лічильники кількості створених з початку роботи об'єктів для вказівки унікальних ідентифікаторів та поточної кількості об'єктів.

#**include** <iostream> //*підключення бібліотеки вводу/виводу*

**using** namespace std; //*застосування простору імен*

**class** CAny //*оголошення класу*

{

**private**: //*закриті члени класу*

//*оголошення статичного поля лічильника загальноїкількості об'єктів*

**static int** tcounter;

//*оголошення статичного поля лічильника поточноїкількості об'єктів*

**static int** ccounter;

**int** ID; //*оголошення поля ідентифікатора об'єкта*

**public**:

CAny() //*оголошення та визначення конструктора*

{

tcounter++; //*інкрементування загального лічильника*

ccounter++; //*інкрементування поточного лічильника*

ID= tcounter; //*визначення ідентифікатора об'єкта*

}

~CAny() //*оголошення та визначення деструктора*

{

ccounter--; //*декрементування поточного лічильника*

}

//*оголошення та визначення методу виводу ідентифікатора*

**void** showID()

{

//*вивід ідентифікатора об'єкта*

**cout**<<"Object ID is "<<ID<<**endl**;

}

//*оголошення та визначення статичного методу виводу поточного лічильника*

**static void** showCounterV()

{

//*вивід значення поточного лічильника*

**cout**<<"Objects: "<<ccounter<<**endl**;

}

};

**int** CAny::ccounter=0; //*визначення статичного поля за умовчуванням*

**int** CAny::tcounter=0; //*визначення статичного поля за умовчуванням*

**void** main()

{

CAny \*pA1,\*pA2; //*оголошення змінних-покажчиків на клас*

pA1=**new** CAny; //*динамічне створення об'єкта*

**delete** pA1; //*вивільнення пам'яті*

pA2=**new** CAny[5]; //*динамічне створення масиву об'єктів*

pA1=**new** CAny; //*динамічне створення об'єкта*

pA1->showID(); //*виклик методу виводу ідентифікатора*

(pA2+3)->showID(); //*виклик методу виводу ідентифікатора*

//*виклик статичного методу виводу поточного лічильника*

CAny::showCounterV();

**delete** pA1; //*вивільнення пам'яті*

**delete** [] pA2; //*вивільнення пам'яті*

//*виклик статичного методу виводу поточного лічильника*

CAny::showCounterV();

}

*У програмі створено клас* CAny*, що включає два статичні поля лічильника; поле унікального ідентифікатора об'єкта; конструктор, що інкрементує лічильники та ініціює змінну ідентифікатор; деструктор, який дектрементує лише лічильник поточної кількості; метод, що виводить на консоль ідентифікатор об'єкта та статичний метод, що відображає на консолі поточну кількість об'єктів. Уголовній функції оголошено дві змінні-покажчика на клас* CAny*. Динамічним шляхом об'єкти створюються та видаляються, на консоль виводяться зазначені ідентифікатори та поточна кількість об'єктів.*

Результат роботи наведеної програми буде полягати у виводі на консоль наступних чотирьох рядків:

Object ID is 7

Object ID is 5

Objects: 6

Objects: 0

Таким чином, завдячуючи загальному лічильнику, кожен об'єкт при створенні отримує унікальний ідентифікатор, відповідає черзі виникнення в процесі виконання програми. Поточний лічильник зберігає кількість об'єктів, що існує на вказаний час.

Треба зазначити, що для першої ініціалізації статичних полів необхідно вказувати глобальне їх визначення перед використанням згідно з наступним форматом:

<*тип даних* ><*ім'я класу* >::<*ім'я поля* >=<з*начення*>;

Таке визначення через операцію глобального дозволу не є новим оголошенням поля. Воно вказує компілятору, що є пам'ять, яка закріплена за зазначеним полем, буде спільною для всіх об'єктів.

Основним призначенням для статичних методів класу є можливість їх виклику незалежно від створених об'єктів через операцію глобального дозволу.

Окрім того, такий підхід не забороняє викликати їх класичним шляхом.

Останній рядок наведеного прикладу реалізує виклик статичного методу showCounterV() при відсутності будь-яких об'єктів класу CAny. Застосування статичних членів класів не залежить від обраного, динамічного чи статичного, шляху створення об'єктів.

**Висновки**

У даному розділи було розглянуто наступні основні питання:

– поліморфізм;

– віртуальні функції;

– абстрактні класи;

– дружні функції та класи;

– статичні члени класів.

**Контрольні питання**

1) Що таке поліморфізм?

2) Яку функцію називають віртуальною?

3) У чому полягає призначення віртуальних функцій?

4) Яка різниця між раннім та пізнім зв'язуванням?

5) Яку віртуальну функцію називають чистою?

6) Для чого потрібні чисті віртуальні функції?

7) Який клас називають абстрактним?

8) Чому при роботі зі спадкоємцем через змінну-покажчик деструктор

базового класу обов'язково повинен бути віртуальним?

9) Як за допомогою віртуальних класів вирішується невизначеність при множинному спадкуванні?

10) Яких правил необхідно дотриматись, щоб реалізувати поліморфізм на основі віртуальних функцій?

11) Які функції називають дружніми?

12) Як вказати, що зазначена функція буде дружньою для класу?

13) Для чого застосовують дружні класи?

14) У чому полягає спостереження при застосування властивостей дружності?

15) Які поля класу називають статичними?

16) У чому полягає сутність застосування статичних полів класу?

17) Як визначити статичне поле за умовчуванням?

18) Які методи класу називають статичними?

19) Як виконується виклик статичних методів класу?

20) Яке основне призначення мають статичні методи класів?

**Лекція 16. Шаблони функцій і класів**

Навчальною метою розділу є ознайомлення читача з механізмом створення шаблонів функцій та класів.

Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

* створювати та використовувати шаблони функцій;
* застосовувати шаблони класів для написання універсальних програмних кодів.

1. **1. Механізм шаблонів**

Під час розробки великих і складних проектів на мові С++, часто виникає необхідність використання функцій, які виконують одну задачу, але використовують дані різних типів.

В мові С для виконання схожої задачі створюється ціла група функцій, імена котрих вказують с даними якого типу працює саме цей екземпляр. Наприклад, для того, щоб отримати модуль числа, необхідно створити наступні функції:

для типу char - cabs()

для типу int - iabs ()

для типу long int - labs()

для типу float - fabs()

для типу double – dabs()

Тіла цих функцій були б повністю ідентичні для аргументу **Х**, що передається:

{**return** (x<0) ? (–x) : x ;} //*повернути значення за модулем*

В мові С++ вирішення цієї задачі має інший вигляд, тому що в цій мові виникає механізм перегруження функцій. Таким чином, була б розроблена така сама група функцій, які мають однакове ім'я abs(). Механізм перегруження спрощує виклик такої функції, але ніяк не розмір програмного коду, який необхідно написати для визначення перегруженних функцій. Наприклад, для знаходження суми двох чисел, необхідно створити функцію sum(), яка має наступні перегруженні визначення:

//*оголошення та визначення функціїрозрахунку суми двох значень типу* ***char***

**double** Sum (**char** a, **char** b)

{**return** a + b ;} //*повернути значення суми*

//*оголошення та визначення функціїрозрахунку суми двох значень типу* ***int***

**double** Sum (**int** a, **int** b)

{**return** a + b ;} //*повернути значення суми*

//*оголошення та визначення функціїрозрахунку суми двох значень типу* ***long int***

**double** Sum (**long int** a, **long int** b)

{**return** a + b ;} //*повернути значення суми*

//*оголошення та визначення функціїрозрахунку суми двох значень типу* ***float***

**double** Sum (**float** a, **float** b)

{**return** a + b ;} //*повернути значення суми*

//*оголошення та визначення функціїрозрахунку суми двох значень типу* ***double***

**double** Sum (**double** a, **double** b)

{**return** a + b ;} //*повернути значення суми*

Як бачимо із наведеного вище прикладу, тіла функцій, які використовуються для виконання певної задачі з даними різних типів, ідентичні. Це можна сприймати і як перевагу, і як недолік. Загалом написання групи таких функцій зводиться до пропису рядків визначення і одного тіла. Далі для всіх визначень, що залишилися, тіло можна просто скопіювати. Однак, що буде якщо тіло функції не настільки просте, як в наведених прикладах, і займає десять або двадцять рядків? Ваша програма під час пропису кожної наступної функції групи буде збільшуватися в розмірах. А що буде, якщо в процесі написання програми ви знайдете помилку в тілі такої функції? Це призведе до того, що її прийдеться виправляти не один раз, а у всіх екземплярах. Таким чином, використання групи функцій для виконання окремої задачі з даними різних типів є не стільки складним, як клопітким процесом.

В мову С++ закладено засіб, який може полегшувати таку задачу. Це механізм застосування шаблонів. Його може бути використано як для функцій, так і для класів.

* + 1. **Шаблон функції**

Оголошення шаблону виконується за наступним форматом:

**template** < **class** <*ім'я шаблонного типу даних* > >

Приклад оголошення шаблону функцій для обчислення різниці двох чисел наведено нижче:

**template** < **class** T > //*зазначення шаблону типу даних T*

T Sub (T a, T b) //*оголошення та визначення функціїрозрахунку різниці*

{

**return** a – b; //*повернути результат розрахунку різниці*

}

Слово **template** є службовим і використовується для оголошення шаблонного типу даних. Ключове слово **class**, взяте у кутові дужки, може бути замінене на **type**.

Коли компілятор зустрічає слово **template**, то практично нічого не виконує. Шаблон не викликає генерацію будь-якого коду, оскільки компілятор ще не може знати, який об'єм пам'яті необхідно виділити для цієї функції.

Шаблон запам'ятовується для наступного використання.

Генерація коду виконується в момент виклику. Виклик такої функції називається реалізація шаблону. При цьому кожен реалізований шаблон називають шаблонною функцією.

Приклад створення і використання шаблона функції для знаходження добутку двох чисел:

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

**using namespace** std; //*застосування простору імен std*

**template** <**class** T> //*зазначення шаблону типу даних T*

//*оголошення та визначення шаблонноїфункції розрахунку добутку*

T Mult (T a, T b)

{

**return** a\*b ; //*повернення значення добутку*

}

**void** main() //*оголошення та визначення головноїфункції*

{

**int** c1 = 3; //*оголошення та визначення цілоїзмінної*

**int** c2 = -7; //*оголошення та визначення цілоїзмінної*

**loat** f1 = 2.3; //*оголошення та визначення змінноїз плаваючою точкою*

**float** f2 = 4.1; //*оголошення та визначення змінноїз плаваючою точкою*

**unsigned int** i1 = 321; //*оголошення та визначення цілоїзмінної*

**unsigned int** i2 = 2; //*оголошення та визначення цілоїзмінної*

//*вивід на консоль результату роботи функціїдля цілих змінних*

**cout** << c1 << '\*' << c2 << '=' << Mult(c1, c2) << **endl** ;

//*вивід на консоль результату роботи функціїдля дійсних змінних*

**cout** << f1 << '\*' << f2 << '=' << Mult(f1, f2) << **endl** ;

//*вивід на консоль результату роботи функціїдля цілих змінних*

**cout** << i1 << '\*' << i2 << '=' << Mult(i1, i2) << **endl** ;

}

Результатом роботи цієї програми будуть наступні рядки:

3 \* (-7) = -21

3 \* 4.1 = 9.43

321 \* 2 = 642

Таким чином, використання шаблону дозволило функції Mult() працювати з будь-яким із базових числових типів (за прикладом з **char**, **float**, **int**).

Під час використання шаблону, об'єм пам'яті, яку займає програма, не змінюється. Але використання шаблонів приводить до зменшення лістинга програми і спрощенню процесу виправлення помилок.

Під час компіляції програми, компілятор визначає який об'єм пам'яті необхідно виділити шаблонній функції на основі типу даних аргументів, що передаються у цю функцію. Таким чином, шаблон функції не є функцією в звичному розумінні. Це скоріше модель, за якою компілятор, зустрічаючи виклик, формує у пам'яті програмний код.

Шаблони функцій можуть мати як один, так і декілька аргументів, що передаються. Причому рішення про те, скільки із них будуть мати шаблонний тип даних, залишається за програмістом. Значення, що повертається, також може і не бути шаблонного типу.

Приклад шаблона функції для знаходження максимального елемента масиву:

**template** <class А> //*зазначення шаблону типу даних А*

//*оголошення та визначення шаблонноїфункції пошуку максимального*

//*елемента масиву*

A GetMaxVal (A \*Arr, **int** n)

{

**int** max = \*Arr; //*оголошення та визначення за умовчуванням змінної*

**for**(**int** i=1; i < n; i++) //*цикл перебору елементів масиву з 1-го*

{

**if** (max < \*(Arr+i)) //*якщо поточний елемент більший за max*

max = \*(Arr+i); //*перевизначити max*

}

**return** max; //*повернути знайдене значення*

}

З прикладу видно, що ім'я шаблонного типу зовсім не має бути однаковим. Як і для змінних значень, імена шаблонних типів можна вибирати будь-які.

В наведеному прикладі шаблонний тип А використовується для одного із двох аргументів і для значення, що повертається.

Необхідно пам'ятати, що для кожної шаблонної функції шаблонний тип даних з вказаним ім'ям заміщує лише один базовий чисельний.

Використовувати шаблонну функцію, як наведено у наступному прикладі, не можна!

Приклад некоректного використання шаблона функції для знаходження добутку двох чисел:

**template** <**class** В> //*зазначення шаблону типу даних В*

//*оголошення та визначення шаблонноїфункції розрахунку добутку*

В Mult(В a, В b)

{**return** a\*b;} //*повернути значення добутку*

**void** main() //*оголошення та визначення головноїфункції*

{

**int** а = 5; //*оголошення та визначення цілоїзмінної*

**float** b = 3.1; //*оголошення та визначення змінноїз плаваючою точкою*

//*вивід результату розрахунку на консоль – хибне застосування шаблону*

// **cout** << a << '\*' << b << '=' << Mult(a, b) << **endl** ;

}

Таке використання шаблону функції Mult() недопустимо, оскільки значення а і b мають різні типи даних, а при визначенні шаблону було вказано, що обидва аргументи мають бути однакового типу.

Та все ж, подібна заборона ще не означає, що задача, в якій необхідно виконувати операції зі значеннями різного типу даних, не має рішення. Просто необхідно використовувати одразу декілька шаблонних типів даних.

Приклад створення і використання шаблону функцій для знаходження в масиві елемента найбільш наближеного до значення, що передається:

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

**using namespace** std; //*застосування простору імен std*

**template** <**class** aT, **class** bT > //*зазначення шаблонів типів даних aTтаbT*

//*оголошення та визначення шаблонноїфункції пошуку близького елемента*

**int** GetValIndex(aT \*Arr, bT val, **int** n)

{

**int** index; //*оголошення цілоїзмінноїдля пошуку індексу*

**float** delta; //*оголошення дійсноїзмінноїдля відхилення*

//*розрахунок відхилення першого елемента від значення*

delta = ((val – \*Arr)<0) ? – (val – \*Arr) : (val – \*Arr);

index = 0; //*визначення змінноїіндексу нулем*

**for**(**int** i = 1; i < n; i ++) //*цикл перебору елементів масиву з 1-го*

//*якщо відхилення поточного менше за знайдене*

**if**(delta > (((val – \*(Arr+i))< 0) ? –(val – \*(Arr+i)) : (val – \*(Arr+i))))

{

//*розрахунок відхилення поточного елемента від значення*

delta = ((val – \*(Arr+i))<0) ? –(val – \*(Arr+i)) : (val – \*(Arr+ i));

index = i; //*визначення змінноїіндексу поточним i*

}

**return** index; //*повернути індекс елемента*

}

**void** main() //*оголошення та визначення головноїфункції*

{

//*оголошення та визначення цілого масиву*

**int** Arr[10] = {1,2,3,4,5,6,7,8,9,0};

//*вивід результату виклику шаблонноїфункціїна консоль*

**cout** << "element N =" << GetValIndex(Arr, 3.721, 10) << **endl**;

}

*Функція дозволяє знайти позицію в масиві числа, яке найбільш відповідає значенню* 3.721*. Таким числом є* 4*, яке в масиві має індекс* 3*. Функція працює одразу з двома шаблонними типами даних* aT *і* bT*.*

Результатом роботи програми буде рядок:

element N = 3

Шаблони функцій можуть бути використані і з типами даних користувачів. Однак у цьому випадку необхідно уважно перевіряти оператори, які використовуються в тілі такої функції для виконання дій над даними, що передаються. Адже будь-який оператор може бути неперевантаженим для типу даних користувача або виконувати будь-яку іншу задачу.

**16.1.2. Шаблон класу**

З шаблонами класів все складається так само, як і з шаблонами функцій. Вони використовуються для розробки моделей класів, які можуть працювати з даними різних типів. Формат оголошення шаблону класу наведено нижче:

**template** < **class** <*ім'я шаблонного типу даних*> >

**class** <*ім'я класу*>

{

<*поля і методи класу*>

}

Тут, як і для шаблонів функцій, ім'я шаблонного типу даних може бути будь-яким. Шаблон класу може включати декілька шаблонних типів даних.

Взагалі, розробка шаблонів класів переслідує дві цілі: створення засобу для збереження даних будь-якого типу і об'єднання функцій для роботи с такими даними в групу. Ці цілі перекликаються з основною задачею, яка переслідується при створенні будь-якого класу.

При роботі з шаблонами класів, на відміну від шаблонів функцій, існує лише одна відмінність: якщо під час виклику шаблонної функції тип даних визначається компілятором автоматично, то при створенні шаблонного об'єкта класу тип даних необхідно чітко вказати.

Приклад створення і використання шаблону класу для виконання найпростіших математичних операцій над масивом базового числового типу:

#**include** <iostream> //*підключення бібліотеки для потокового вводу/виводу*

**using namespace** std; //*застосування простору імен std*

**template** <**class** T> //*зазначення шаблону типу даних*

**class** CMath //*оголошення класу CMath*

{

**public**: //*відкриті члени класу*

//*оголошення та визначення шаблонноїфункції отримання суми елементів*

//*масиву*

T GetSum(T \* Arr, int n)

{

T S = 0; //*оголошення змінноїдля збереження суми значень*

**for** (**int** i = 0; i < n; i ++) //*цикл перебору елементів масиву*

S += \*(Arr + i); //*додавання до суми*

**return** S; //*повернення отриманоїсуми*

}

//*оголошення та визначення шаблонноїфункції отримання середнього*

//*арифметичного значення елементів масиву*

**double** GetMidVal(T \*Arr, **int** n)

{

//*оголошення змінноїдля збереження суми значень*

**double** S = 0;

**for** (**int** i = 0; i < n; i ++) //*цикл перебору елементів масиву*

S += \*(Arr + i); //*додавання до суми*

**return** S/n; //*повернення отриманого значення*

}

};

**void** main() //*оголошення та визначення головноїфункції*

{

//*оголошення та визначення масиву цілих чисел*

**int** iMas[5] = {3, 173, 370, 14, -100};

//*оголошення та визначення масиву дійсних чисел*

**float** fMas[5] = {7.4, 12.72, -7.11, 0.36, -30.8};

CMath <**int**> im; //*оголошення шаблонного об'єкта типу* ***int***

//*вивід на консоль суми елементів масиву цілих чисел*

**cout** << "Sum of iMas elements is:" << im.GetSum(iMas, 5) <<**endl**;

//*вивід на консоль середнього арифметичного значенння елементів масиву iMas*

**cout** << "Mid value of iMas elements is:" << im.GetMidVal(iMas, 5) <<**endl**;

CMath <**float**> fm; //*оголошення шаблонного об'єкта типу* ***float***

//*вивід на консоль суми елементів масиву дійсних чисел*

**cout** << "Sum of fMas elements is:" << fm.GetSum(fMas, 5) <<**endl**;

//*вивід на консоль середнього арифметичного значення елементів масиву fMas*

**cout** << "Mid value of fMas elements is:" << fm.GetMidVal(fMas, 5) <<**endl**;

}

Результатом виконання цієї програми будуть наступні рядки:

Sum of iMas elements is: 460

Mid value of iMas elements is: 92

Sum of fMas elements is: -17.43

Mid value of fMas elements is: -3.486

З прикладу видно, що тип даних, з яким буде використано шаблон класу для формування в пам'яті шаблонного об'єкта визначається під час оголошення об'єкта в кутових дужках між ім'ям класу та ім'ям об'єкта за наступним форматом:

<*ім'я класу*> < <*ім'я типу даних для створення шаблонного об'єкта*> > <*ім'я об'єкта класу*>;

Рядок оголошення шаблонного об'єкта вказує не лише на виділення пам'яті для усіх полів цього об'єкта класу, а також на створення прототипів длят всіх методів класу, які використовують шаблонний тип даних.

Коли визначення методів здійснюється за рамками шаблону класу, необхідно вказати шаблонний тип даних для кожної функції і ідентифікувати при цьому клас, що їх включає, як шаблон.

Приклад створення і використання шаблону класу для формування стека і виконання над ним простих операцій:

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

**using namespace** std; //*застосування простору імен std*

**template** <**class** T> //*зазначення шаблону типу даних T*

**class** Stack //*оголошення класу реалізаціїстека*

{

**private**: //*закриті члени класу*

**int** count; //*оголошення змінноїрахування числа елементів в стеку*

**int** Max; //*оголошення змінноїдля задання розмірності стека*

T \*Arr; //*оголошення змінної-покажчика для динамічного массиву*

**public**: //*відкриті члени класу*

//*оголошення та визначення конструктора класу*

Stack(**int** n) : Max(n) //*ініціалізація Max за допомогою аргументу n*

{

//*створення динамічного масиву на Max елементів*

Arr = **new** T[Max];

count = 0; //*визначення змінної – 0 елементів*

}

**bool** Push(T val); //*оголошення методу додавання значення у стек*

**bool** Pop(T& var); //*оголошення методу вилучення значення зі стека*

**void** Clear(); //*оголошення методу очищення стека*

//*оголошення методу отримання кількості елементів стека*

**int** GetCount();

//*оголошення методу отримання максимального значення стека*

**bool** GetMaxVal(T& max);

//*оголошення методу отримання мінімального значення стека*

**bool** GetMinVal(T& min);

~Stack (); //*оголошення деструктора класу*

};

//*визначення методу додавання значення у стек*

**template** <**class** T> //*зазначення шаблону типу даних T*

**bool** Stack <T>::Push(T val)

{

**if** (count != Max) //*якщо в стеку є вільне місце*

{

\*(Arr + count) = val; //*записати у стек значення*

count++; //*збільшити значення лічильника елементів*

**return** true; //*повернути true*

}

**else** //*якщо в стеку немає вільного місця*

**return** false; //*повернути false*

}

//*визначення методу вилучення значення зі стека*

**template** <**class** T> //*зазначення шаблону типу даних T*

**bool** Stack <T>::Pop(T& var)

{

**if** (count == 0) //*якщо стек порожній*

**return** false; //*повернути false*

**else** //*якщо стек не порожній*

{

count--; //*зменшити значення рахунку елементів*

var = \*(Arr + count); //*записати в var значення*

**return** true; //*повернути true*

}

}

//*визначення методу очищення стека*

**template** <**class** T> //*зазначення шаблону типу даних T*

**void** Stack <T>::Clear()

{

count = 0; //*визначити лічильник нулем*

}

//*визначення методу отримання кількості елементів стека*

**template** <**class** T> //*зазначення шаблону типу даних T*

**int** Stack <T>::GetCount()

{

**return** count; //*повернути значення лічильника елементів*

}

//*визначення методу отримання максимального значення стека*

**template** <**class** T> //*зазначення шаблону типу даних T*

**bool** Stack <T>::GetMaxVal(T& max)

{

**if** (count == 0) //*якщо стек порожній*

**return** false; //*повернути false*

**else** //*якщо стек не порожній*

{

max = \*Arr; //*присвоїти max значення першого елемента стека*

//*перебрати елементи стека, починаючи з 2-го*

**for** (**int** i =1; i < count; i ++)

//*якщо значення поточного елемента більше max*

**if** (max < \*(Arr + i))

max = \*(Arr + i); //*перевизначити значення max*

**return** true; // *повернути true*

}

}

//*визначення методу отримання мінімального значення стека*

**template** < **class** T > //*зазначення шаблону типу даних T*

**bool** Stack <T>::GetMinVal(T& min)

{

**if** (count == 0) //*якщо стек порожній*

**eturn** false; //*повернути false*

**else** //*якщо стек не порожній*

{

min = \*Arr; //*присвоїти min значення першого елемента стека*

//*перебрати елементи стека, починаючи з 2-го*

**for** (**int** i =1; i < count; i ++)

//*якщо значення наявного елемента меньше за min*

**if** (min > \*(Arr + i))

min = \*(Arr + i); //*перевизначити значення min*

**return** true; //*повернути true*

}

}

//*визначення деструктора класу*

**template** <**class** T> //*зазначення шаблону типу даних T*

Stack <T> :: ~Stack()

{

**delete** [] Arr; //*видалення динамічного масиву*

}

**void** main() //*оголошення та визначення головноїфункції*

{

//*оголошення шаблонного об'єкта is цілого типу і створення стека на 10*

//*елементів*

Stack <**int**> is(10);

//*оголошення шаблонного об'єкта fs дійсного типу і створення стека на 5*

//*елементів*

Stack <**float**> fs(5);

//*оголошення масиву цілих чисел*

**int** iMas[15] = {3, 5, 7, -4, 0, -12, 82, 100, -31, 300, 27, 42, -13, -29, -100};

//*оголошення масиву дійсних чисел*

**float** fMas[7] = {3.3, 7.9, -3.4, 15.01, 9.6, 0.17, 10.7};

**int** i =0; //*оголошення та визначення змінноїдля перебору елементів*

**while** (is.Push (iMas [i++])) //*цикл заповнення стека is*

{ }

i = 0; //*визначення змінноїперебору елементів нулем*

**while** (fs.Push (fMas [i++])) //*цикл заповнення стека fs*

{ }

**int** v; //*оголошення змінноїцілого типу для збереження значення зі стека*

**if** (is.Pop (v)) //*якщо стек не порожній, то зчитати значення в v*

//*вивід зчитаного значення на консоль*

**cout** << "INT Stack last value was: " << v << **endl**

//*вивід на консоль числа елементів у стеку цілого типу*

**cout** << "In INT Stack are " << is.GetCount () << "values" << **endl**;

//*вивід на консоль числа елементів у стеку дійсного типу*

**cout** << "In FLOAT Stack are " << fs.GetCount () << "values" << **endl**;

//*оголошення змінноїцілого типу для максимального значення стека is*

**int** imax;

//*якщо стек не порожній, то отримати в imax максимальне значення*

**if** (is.GetMaxVal (imax))

//*вивід на консоль максимального значення стека is*

**cout** << "Max value of INT Stack is: " << imax << **endl**;

//*оголошення змінноїдійсного типу для мінімального значення стека fs*

**float** fmin;

//*якщо стек не порожній, то отримати в fmin мінімальне значення*

**if** (fs.GetMinVal (fmin))

//*вивід на консоль мінімального значення стека fs*

**cout** << "Min value of FLOAT Stack is: " << fmin << **endl**;

}

Результатом виконання даної програми будуть наступні рядки:

INT Stack last value was: 300

INT Stack last value was: -31

In INT Stack are 8 values

In FLOAT Stack are 5 values

Max value of INT Stack is: 100

Min value of FLOAT Stack is: -3.4

Перші два рядки характеризують коректність роботи стека. Виклик втциклі метода Push() приводить до перезапису із iMas в стек типу цілих чисел 10 значень (по 300), а із fMas в стек типу дійсних чисел 5 значень (по 9.6). Потім, два виклики метода Рор() для стека типу цілих чисел вилучають два останніх значення (300 та -31).

Третій та четвертий рядок вказують скільки значень в кожному із стеків (8 в типі цілих чисел і 5 в типі дійсних чисел). Останні рядки виводять на консоль для стека типу цілих чисел максимальне, а для стека типу дійсних чисел – мінімальне значення.

Як видно з прикладу, для того, щоб показати, що функція є методом шаблона класу, необхідно дотримуватись наступного формату:

**template** < **class**<*ім'я шаблонного типа даних*>>

<*тип даних*> <*ім'я шаблону класу*> <<*ім'я шаблонного типу даних*>> :: <*ім'я*  
*функції*> (<*список аргументів*>)

{

<*тіло функції*>

}

В основному шаблони класів використовуються для створення так званихтконтейнерів для збереження даних (складно організованих масивів). Для мови С++ спеціально розроблена допоміжна бібліотека STL (Standart Template Library – стандартна бібліотека шаблонів), яка містить основні види контейнерів та засоби роботи з ними.

Під час створення шаблонів функцій і класів бажано виконувати спочатку розробку простої функції або класу, що базуються на певному типі даних, а потім, після їх виправлення та тестування, виконувати перетворення в шаблон.

**Висновки**

У даному розділи було розглянуто наступні основні питання:

– застосування шаблонів типів класів;

– шаблони функцій;

– шаблони класів.

**Контрольні питання**

1) Що називають шаблоном?

2) Для чого застосовуються шаблони?

3) За допомогою якого службового слова програми створюється шаблон типу даних?

4) Які шаблони є найбільш поширеними?

5) Наведіть формат оголошення шаблонної функції.

6) Яке службове слово замість **class** можна використовувати для створення шаблонного типу?

7) Як називають виклик шаблонної функції?

8) У чому різниця між звичайними та шаблонними функціями відносно компілятора?

9) Скільки аргументів може мати шаблонна функція?

**Тема 4. Програмування динамічних структур**

**Лекція 17. Алгоритми впорядкування**

Навчальною метою розділу є ознайомлення читача з основними алгоритмами, що застосовуються для впорядковування послідовностей даних.

Внаслідок вивчення матеріалу даного розділу читач повинен знати:

– метод бульбашкового сортування;

– метод сортування Шейкера;

– метод сортування вибіркою;

– метод сортування вставкою;

– метод сортування вибіркою-вставкою;

– метод сортування Шелла;

– метод швидкого сортування;

– перевагу, що надає впорядкованість при застосуванні алгоритмів пошуку.

**17.1. Методи сортування**

При роботі з масивами неврегульованих значень для програміста важливу роль грає знання методик сортування. Сортування дозволяє упорядковувати значення послідовностей і на основі цього зменшує час пошуку елементів в них.

Сортування – це процес поетапного впорядковування значень елементів масиву за убуванням або за зростанням.

Прохід впорядковування – це перебір елементів послідовності від одного краю до іншого.

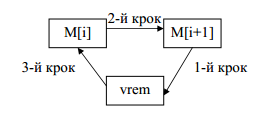
Кроком сортування називають етап проходу впорядковування, під час якого опрацьовується один елемент послідовності.

Існує безліч методик сортування. Найпростішими (найбільший час роботи) є сортування методами обміну, вибірки і вставки.

**17.1.1. Сортування обміном**

Сортування обміном – це, коли впорядковування елементів послідовності виконується за допомогою обміну значень між елементами.

Принцип обміну двох елементів масиву значеннями засновано на схемі, наведеній на рис. 17.1.



***Рис. 17.1. Схема обміну елементів значеннями***

Спочатку створюється допоміжна змінна. Значення одного з елементів переписується у неї (1-й крок). Потім у цей елемент перезаписується значення з іншого елемента (2-й крок). І на останньому етапі значення з допоміжної змінної переприсвоюється у другий елемент (3-й крок).

Неоптимізований метод бульбашкового сортування здійснюється шляхом обміну. Його засновано на виконанні для неввпорядкованої послідовності значень N-1 проходів по N-1 кроків (N – число елементів послідовності). На кожному кроці виконується порівняння значення поточного елемента із значенням наступного і їх розташування в потрібному порядку шляхом обміну.

За (N-1)2 кроків можна виконати повне впорядковування будь-якої послідовності. Схематичний приклад сортування за бульбашковим методом цілочисельного масиву від більшого до меншого наведено нижче:

масив: -3 12 3 4 21

1-й прохід 1-й крок 12 -3 3 4 21

1-й прохід 2-й крок 12 3 -3 4 21

1-й прохід 3-й крок 12 3 4 -3 21

1-й прохід 4-й крок 12 3 4 21 -3

2-й прохід 1-й крок 12 3 4 21 -3

2-й прохід 2-й крок 12 4 3 21 -3

2-й прохід 3-й крок 12 4 21 3 -3

2-й прохід 4-й крок 12 4 21 3 -3

3-й прохід 1-й крок 12 4 21 3 -3

3-й прохід 2-й крок 12 21 4 3 -3

3-й прохід 3-й крок 12 21 4 3 -3

3-й прохід 4-й крок 12 21 4 3 -3

4-й прохід 1-й крок 21 12 4 3 -3

4-й прохід 2-й крок 21 12 4 3 -3

4-й прохід 3-й крок 21 12 4 3 -3

4-й прохід 4-й крок 21 12 4 3 -3

Оптимізований метод бульбашкового сортування відрізняється від неоптимізованого тим, що після кожного проходу число елементів, що піддаються подальшому порівнянню, зменшується на один. Оптимізація заснована на тому, що за один прохід з будь-якої позиції послідовності максимальне або мінімальне значення переміщується на крайню, таким чином, при наступному проході значення цього елементу обміну не підлягає.

Схематичний приклад сортування за оптимізованим бульбашковим методом цілочисельного масиву від більшого до меншого наведено нижче:

масив: -3 12 3 4 21

1-й прохід 1-й крок 12 -3 3 4 21

1-й прохід 2-й крок 12 3 -3 4 21

1-й прохід 3-й крок 12 3 4 -3 21

1-й прохід 4-й крок 12 3 4 21 -3

2-й прохід 1-й крок 12 3 4 21 -3

2-й прохід 2-й крок 12 4 3 21 -3

2-й прохід 3-й крок 12 4 21 3 -3

3-й прохід 1-й крок 12 4 21 3 -3

3-й прохід 2-й крок 12 21 4 3 -3

4-й прохід 1-й крок 21 12 4 3 -3

Метод сортування Шейкера є вдосконаленням оптимізованого бульбашкового сортування і ґрунтується на тому, що кожен наступний прохід виконується із зміною напряму вичитування елементів послідовності. Завдяки цьому протягом двох проходів на відповідних позиціях виявляються як максимальне, так і мінімальне значення послідовності.

Схематичний приклад сортування за методом Шейкера цілочисельного масиву від більшого до меншого наведено нижче:

масив: -3 12 3 4 21

1-й прохід 1-й крок 12 -3 3 4 21

1-й прохід 2-й крок 12 3 -3 4 21

1-й прохід 3-й крок 12 3 4 -3 21

1-й прохід 4-й крок 12 3 4 21 -3

2-й прохід 1-й крок 12 3 21 4 -3

2-й прохід 2-й крок 12 21 3 4 -3

2-й прохід 3-й крок 21 12 3 4 -3

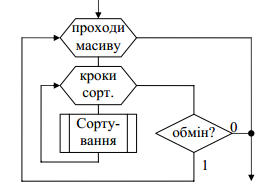
3-й прохід 1-й крок 21 12 3 4 -3

3-й прохід 2-й крок 21 12 4 3 -3

4-й прохід 1-й крок 21 12 4 3 -3

Оптимізоване бульбашкове сортування і сортування Шейкера можуть бути покращені за допомогою перевірки кількості виконаних обмінів у кінці кожного проходу і зупинці сортування при їх нульовому числі. Підхід ґрунтується на тому, що під час сортування обміном з кроком в один елемент при нульовому числі обмінів за прохід, послідовність є повністю впорядкованою і подальші проходи лише збільшують час виконання сортування.

Схематичне відображення зупинення сортування при нульовому числі обмінів протягом проходу наведено у вигляді схеми алгоритму на рис. 17.2.



***Рис. 17.2. Схема алгоритму рахування числа обмінів за прохід***

Схематичний приклад сортування за оптимізованим бульбашковим методом цілочисельного масиву від більшого до меншого з рахуванням кількості обмінів за прохід наведено нижче:

масив: -3 12 3 6 5

k=0 →

1-й прохід

1-й прохід 1-й крок 12 -3 3 6 5

k=1

1-й прохід 2-й крок 12 3 -3 6 5

k=2

1-й прохід 3-й крок 12 3 6 -3 5

k=3

1-й прохід 4-й крок 12 3 6 5 -3

k=4

k≠0, k=0 → 2-й прохід

2-й прохід 1-й крок 12 3 6 5 -3

k=0

2-й прохід 2-й крок 12 6 3 5 -3

k=1

2-й прохід 3-й крок 12 6 5 3 -3

k=2

k≠0, k=0 → 3-й прохід

3-й прохід 1-й крок 12 6 5 3 -3

k=0

3-й прохід 2-й крок 12 6 5 3 -3

k=0

k=0, зупинення

Схематичний приклад сортування за методом Шейкера цілочисельного масиву від більшого до меншого з рахуванням кількості обмінів за прохід наведено нижче:

масив: -3 12 4 3 21

k=0 →

1-й прохід

1-й прохід 1-й крок 12 -3 4 3 21

k=1

1-й прохід 2-й крок 12 4 -3 3 21

k=2

1-й прохід 3-й крок 12 4 3 -3 21

k=3

1-й прохід 4-й крок 12 4 3 21 -3

k=4

k≠0, k=0 →

2-й прохід

2-й прохід 1-й крок 12 4 21 3 -3

k=1

2-й прохід 2-й крок 12 21 4 3 -3

k=2

2-й прохід 3-й крок 21 12 4 3 -3

k=3

k≠0, k=0 → 3-й прохід

3-й прохід 1-й крок 21 12 4 3 -3

k=0

3-й прохід 2-й крок 21 12 4 3 -3

k=0

k=0, зупинення

**17.1.2. Сортування вибіркою**

Сортування вибіркою – це, коли вся послідовність умовно розбивається на впорядковану і невпорядковану множини, причому з кожним кроком число впорядкованих елементів збільшується на одиницю, починаючи з нуля. На кожному кроці проводиться пошук максимального або мінімального, залежно від напряму впорядковування, значення серед невпорядкованої множини і обмін його значення з елементом на поточній позиції, який після обміну стає останнім значенням у впорядкованій множині.

Схематичний приклад сортування методом вибірки від більшого до меншого наведено нижче:

масив: -3 21 3 4 12

1-й крок 21 -3 3 4 12

2-й крок 21 12 3 4 -3

3-й крок 21 12 4 3 -3

4-й крок 21 12 4 3 -3

**17.1.3. Сортування вставкою**

Сортування вставкою засновано на послідовному переборі невпорядкованих елементів і виконанні вставки поточного елемента на визначену правилами сортування позицію серед вже впорядкованих значень.

Всі елементи, які виявляються меншими (більшими) за поточне значення, переміщуються на одну позицію в напрямку проходу сортування до колишньої позиції поточного елемента.

Схематичний приклад сортування методом вставки від більшого до меншого наведено нижче:

масив: -3 21 3 12 4

1-й крок 21 -3 3 12 4

2-й крок 21 3 -3 12 4

3-й крок 21 12 3 -3 4

4-й крок 21 12 4 3 -3

**17.1.4. Сортування вибіркою-вставкою**

Метод сортування вибіркою-вставкою є результатом синтезу двох простих сортувань – вибірки і вставки. Його засновано на аналогічній методиці вибірки: розбиття послідовності на дві множини, проте замість обміну значеннями між знайденим елементом і поточним виконується вставка його значення на відведену йому правилом сортування позицію у вже впорядкованій множині, при цьому виконується переміщення значень елементів масиву на 1 крок.

Схематичний приклад сортування методом вставки від більшого до меншого наведено нижче:

масив: -3 21 3 12 4

1-й крок 21 -3 3 12 4

2-й крок 21 12 -3 3 4

3-й крок 21 12 4 -3 3

4-й крок 21 12 4 3 -3

**2.1.5. Поліпшені методи сортування**

Серед поліпшених методів сортування виділяють метод Шелла і швидке сортування, як найчастіше вживані і такі, що відрізняються відносно малим часом впорядковування.

Під час сортування за методом Шелла для кожного проходу обирається свій унікальний крок (не перевищує значення N-1, де N – число елементів масиву). Поточний елемент послідовності порівнюється з елементом, який відрізняється від нього на крок поточного проходу і при невідповідності їх порядку виконується обмін. З математичної точки зору слід уникати кроків за степенями значення два, проте, обов'язково повинен бути присутнім крок із значенням один. Для більшої ефективності сортування значення кроків для двох послідовних проходів (неостанніх) повинно відрізнятися більш, ніж на одиницю. Зазвичай максимальний крок обирається більшим чи таким, що дорівнює значенню N/2.

Схематичний приклад сортування методом Шелла від більшого до меншого наведено нижче:

масив: -3 12 3 4 21

1-й прохід 1-й крок 4 12 3 -3 21

1-й прохід 2-й крок 4 21 3 -3 12

2-й прохід 1-й крок 4 21 3 -3 12

2-й прохід 2-й крок 4 21 3 -3 12

2-й прохід 3-й крок 4 21 12 -3 3

3-й прохід 1-й крок 21 4 12 -3 3

3-й прохід 2-й крок 21 12 4 -3 3

3-й прохід 3-й крок 21 12 4 -3 3

3-й прохід 4-й крок 21 12 4 3 -3

Швидке сортування – найбільш ефективна методика, яка доволі часто застосовується у системному програмному забезпеченні. В основі цього сортування лежить рекурсивний підхід. Сортування базується на виборі для послідовності неврегульованих значень елемента, значення якого виступатиме в ролі компаранда (зазвичай близьке до середнього арифметичного значення елементів масиву). Інші елементи розміщуються щодо компаранда наступним чином – великі з одного боку, а менші з іншого. Після першого проходу та сама операція виконується з послідовностями значень справа та зліва від компаранда. Таким чином на кожному проході число компарандів для всієї послідовності дорівнює 2(N-1) (N – число елементів масиву). Операції виконуються до тих пір, поки хоча б одна послідовність містить більше ніж два елементи.

Схематичний приклад швидкого сортування від більшого до меншого наведено нижче:

масив: -3 12 3 4 21

компаранд = 4

1-й прохід 1-й крок 12 3 4 21 -3

1-й прохід 2-й крок 12 4 21 -3 3

1-й прохід 3-й крок 21 12 4 -3 3

компаранди = 12, 3

2-й прохід 1-й крок 21 12 4 3 -3

**17.2. Застосування алгоритмів впорядкування**

Основним призначенням алгоритмів впорядкування є оптимізація пошуку елементів у послідовностях.

Для розуміння розглянемо два алгоритми пошуку, перший застосовується до невпорядкованих послідовностей і називається лінійним, а другий, для впорядкованих, двійковим.

Сутність лінійного пошуку полягає у послідовному переборі всіх елементів невпорядкованої послідовності до тих пір, поки значення не буде знайдено.

Схематичний приклад лінійного пошуку наведено нижче:

Пошук значення 5 послідовність

Крок 1, N≠1 1

Крок 2, N≠7 7

Крок 3, N≠12 12

Крок 4, N≠8 8

Крок 5, N≠3 3

Крок 6, N≠9 9

Крок 7, N≠10 10

Крок 8, N≠34 34

Крок 9, N≠-2 -2

Крок 10, N=5, зупинення 5 11

Двійковий пошук засновано на бінарних запитах до впорядкованої послідовності. Запит виконується до серединного елемента множини елементів послідовності. На запит у якості результату отримується наступні значення:

0 – значення менше за серединний елемент;

1 – значення більше за серединний елемент.

Кожен наступний запит зменшує область пошуку вдвічі.

Схематичний приклад двійкового пошуку наведено нижче:

Пошук значення 5

Послідовність

-2

1

Крок 2, N>3 3

Крок 4, N=5, зупинення 5

Крок 3, N<8 8

Крок 1, N<9 9

10

11

12

34

Таким чином, впорядкованість послідовності дозволяє застосовувати двійковий пошук, що у декілька разів зменшує час пошуку вказаного елемента.

**Висновки**

У даному розділи було розглянуто наступні основні питання:

– прості методики сортування;

– поліпшені методики сортування;

– застосування впорядкування для зменшення часу на пошук.

**Контрольні питання**

1) Дайте визначення поняттю сортування.

2) Які є основні види сортування?

3) Як реалізується неоптимізований метод бульбашкового сортування?

4) Скільки порівнянь виконується під час неоптимізованого методу бульбашкового сортування?

5) У чому полягає оптимізація методу бульбашкового сортування?

6) Чим метод сортування Шейкера відрізняється від методу бульбашкового сортування?

7) Для чого при сортуванні методами обміну може знадобитися рахування кількості обмінів за прохід?

8) Як виконується обмін значеннями між двома елементами масиву?

9) У чому полягає сутність сортування вибіркою?

10) Як виконується сортування за методом вставки?

11) Чим відрізняються між собою методи сортування вставкою та вибіркою-вставкою?

12) У чому полягає сутність сортування методом Шелла?

**Лекція** **18**. Клас роботи з рядками

Навчальною метою розділу є ознайомлення читача з роботою із рядками на основі спеціального класу мови С++.

Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

– застосовувати клас роботи з рядками в програмі;

– виконувати операції з об'єктами класу;

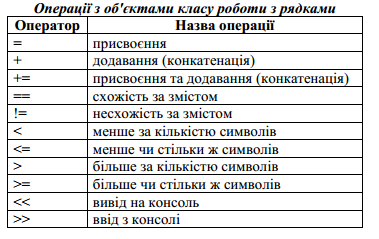
– застосовувати методи класу.

**18.1. Клас string**

У мові С++ для роботи з текстовою інформацією застосовується спеціальний клас, опис якого знаходиться у заголовному файлі "string.h". Окрім його підключення у програмі, треба також зазначити простір імен std. Методи класу будуть доступні крізь його об'єкти. Клас string дозволяє виконувати зі своїми об'єктами стандартні операції присвоєння, додавання та порівняння.

Таким чином, якщо треба поєднати декілька рядків у один, достатньо скористатися оператором "+". Нижче наведено табл. 3.1 з операторами, які можна використовувати при роботі з об'єктами класу string.

Таблиця 18.1

******

Розглянемо для прикладу програму, що виводить на консоль декілька рядків, що є значеннями об'єктів класу роботи з рядками:

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

#**include** <string> //*підключення бібліотеки "string.h"*

**using namespace** std; //*застосування простору імен std*

**void** main()

{

string s, s1, s2; //*оголошення об'єктів класу string s, s1, s2*

string s3("!!!"); //*оголошення та визначення об'єкта s3*

s1="This year is a very "; //*визначення змінної s1*

s2="important for me"; //*визначення змінної s2*

s=s1+s2+s3; //*додавання рядків*

**cout**<<s<<"\r\n"<<**endl**; //*вивід результату на консоль*

s1+=s2+s3; //*додавання рядків*

**cout**<<s1<<"\r\n"<<**endl**; //*вивід результату на консоль*

//*порівняння рядків за кількістю символів і вивід результату на консоль*

**cout**<<(s1>s2)<<"\r\n"<<**endl**;

//*порівняння рядків за змістом символіввивід результату на консоль*

**cout**<<(s2= =s3)<<"\r\n"<<**endl**;

}

Після виконання наведеного коду на консоль буде виведено наступні рядки:

This year is a very important for me!!!

This year is a very important for me!!!

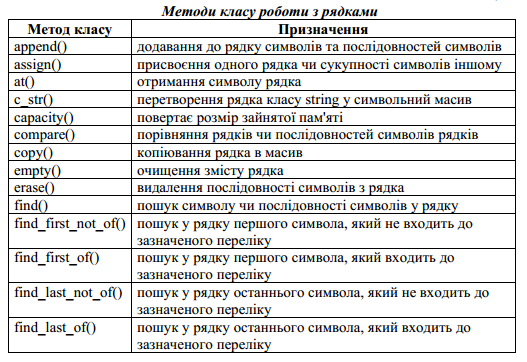
1 (*рядок s1 більше s2*)

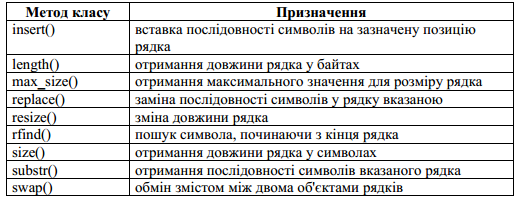
0 (*рядок s2 відрізняється від s3 по символам*)

Основними методами для роботи з рядками, що поєднує у собі клас

"string", є наведені у табл. 18.2.

Таблиця 18.2

******

**** Спрощені формати та приклади роботи з цими методами наведено нижче.

string append(string str, **int** start, **int** num)

*Аргументи:*

*str – рядок, символи якого буде додано;*

*start – номер першого символа послідовності у рядку str;*

*num – число символів послідовності.*

*Значення, що повертається: отриманий рядок.*

string append(**int** count, string str)

*Аргументи:*

*count – кількість повторів для доданих символів;*

*str – послідовність символів для додавання.*

*Значення, що повертається: отриманий рядок.*

Приклад застосування:

string s1,s2; //*оголошення двох об'єктів класу роботи з рядками*

s1="It's a beautiful"; //*визначення об'єкта константним рядком*

s2="Only one day"; //*визначення об'єкта константним рядком*

s1.append(s2,8,4); //*доповнити перший об'єкт підрядком з другого*

s1.append(3,'!'); //*доповнити перший об'єкт символом '!'*

**cout**<<s1<<**endl**; //*вивести результат на консоль*

Результат роботи, що буде виведено на консоль:

It's a beautiful day!!!;

string assign(string str, **int** start, **int** num)

*Аргументи:*

*str – рядок, послідовність символів якого буде присвоєно;*

*start – номер першого символа послідовності у рядку str;*

*num – кількість символів.*

*Значення, що повертається: отриманий рядок.*

**char** at(**int** num)

*Аргументи:*

*num – номер символа, що повертається.*

*Значення, що повертається: символ, що знаходиться у рядку на позиції num.*

**const char** \*c\_str()

*Аргументи: відсутні.*

*Значення, що повертається: покажчик на перший елемент масиву символів.*

**int** capacity()

*Аргументи: відсутні.*

*Значення, що повертається: розмір зайнятоїоб'єктом пам'яті.*

**int** compare(**int** start, **int** num, string str)

*Аргументи:*

*start – позиція першого символа послідовності для порівняння;*

*num – кількість символів для порівняння;*

*str – рядок, з яким буде відбуватися порівняння.*

*Значення, що повертається: -1 – якщо вибрана послідовність за довжиною коротше ніж str; 1 – якщо вибрана послідовність довша ніж str; 0 – якщо вони є такими, що дорівнюються.*

Приклад застосування:

string s1,s2,s3,s4; //*оголошення чотирьох об'єктів класу роботи з рядками*

s1="123456789"; //*визначення об'єкта числовим рядком*

s2="123456789"; //*визначення об'єкта числовим рядком*

s3="1234567890"; //*визначення об'єкта числовим рядком*

s4="12345678"; //*визначення об'єкта числовим рядком*

//*вивід на консоль результату порівняння*

**cout**<<s1.compare(0,9,s2)<<"\r\n"<<**endl**;

//*вивід на консоль результату порівняння*

**cout**<<s1.compare(0,9,s3)<<"\r\n"<<**endl**;

//*вивід на консоль результату порівняння*

**cout**<<s1.compare(0,9,s4)<<"\r\n"<<**endl**;

Результат роботи, що буде виведено на консоль:

0

-1

1

**int** copy(**char** \*ch, **int** num, **int** start)

*Аргументи:*

*ch – покажчик на перший елемент масиву для запису послідовності з рядка;*

*num – кількість символів, що буде скопійовано;*

*start – номер початкового символа послідовності.*

*Значення, що повертається: кількість символів, що насправді було скопійовано.*

**bool** empty()

*Аргументи: відсутні.*

*Значення, що повертається: true – якщо рядок було очищено; false – якщо рядок ще має символи.*

string erase(**int** start=0, **int** num=-1)

*Аргументи:*

*start – номер першого символа, що буде видалено;*

*num – кількість символів для видалення.*

*Значення, що повертається: отриманий рядок.*

**int** find(string str, **int** start=0)

*Аргументи:*

*str – символ, чи послідовність символів для пошуку;*

*start – позиція у рядку, з якоїтреба почати пошук.*

*Значення, що повертається: позиція першого символа послідовності пошуку у рядку.*

**int** find\_first\_not\_of(string str, **int** start=0)

*Аргументи:*

*str – символ чи сукупність символів для пошуку;*

*start – позиція у рядку, з якоїтреба почати пошук.*

*Значення, що повертається: позиція першого символа у рядку, який не входить до заданоїпослідовності.*

**int** find\_first\_of(string str, **int** start=0)

*Аргументи:*

*str – символ чи сукупність символів для пошуку;*

*start – позиція у рядку, з якоїтреба почати пошук.*

*Значення, що повертається: позиція першого символа у рядку, який входить до заданої послідовності.*

**int** find\_last\_not\_of(string str, **int** start=0);

*Аргументи:*

*str – символ чи сукупність символів для пошуку;*

*start – позиція у рядку, з якоїтреба почати пошук.*

*Значення, що повертається: позиція останнього символа у рядку, який не входить до заданоїпослідовності.*

**int** find\_last\_of(string str, **int** start=0)

*Аргументи:*

*str – символ чи сукупність символів для пошуку;*

*start – позиція у рядку, з якоїтреба почати пошук.*

*Значення, що повертається: позиція останнього символа у рядку, який входить до заданоїпослідовності.*

Приклад застосування:

string s="See also synchronous verb completion.";

**cout**<<s.find('e')<<"\r\n"<<**endl**;

**cout**<<s.find\_first\_of("hak")<<"\r\n"<<**endl**;

**cout**<<s.find\_first\_not\_of("esa")<<"\r\n"<<**endl**;

**cout**<<s.find\_last\_of("fop")<<"\r\n"<<**endl**;

**cout**<<s.find\_last\_not\_of("fop")<<"\r\n"<<**endl**;

**cout**<<s.rfind('o')<<"\r\n"<<**endl**;

Результат роботи, що буде виведено на консоль:

1 (*символ 'е' знайдено на 1-й позиції*)

4 (*символ 'а', який є у сукупності "hak", знайдено на 4-й позиції*)

0 (*символ 'S', якого немає у сукупності "esa", знайдено на 0-й позиції*)

34 (*символ 'о', який є у сукупності "fop", знайдено на 34-й позиції*)

36 (*символ '.', якого немає у сукупності "fop", знайдено на 36-й позиції*)

34 (*символ 'о' знайдено на 34-й позиції*)

string insert(**int** pos, string str)

*Аргументи:*

*pos – позиція, з якоїпочнеться вставка послідовності символів;*

*str – послідовність символів для вставки.*

*Значення, що повертається: отриманий рядок.*

**int** length()

*Аргументи: відсутні.*

*Значення, що повертається: довжина рядка у байтах.*

Приклад застосування:

string s1; //*оголошення об'єкта класу роботи з рядками*

s1="Today We have a lot of news"; //*визначення об'єкта константним рядком*

**for** (**int** i=0; i<s1.length(); i++) //*цикл перебору всіх символів рядка*

{

**if** (s1.at(i)=='a') //*якщо поточний символ є 'a'*

**cout**<<"pos = "<<i<<"\r\n"<<**endl**; //*вивести номер позиції*

}

Результат роботи, що буде виведено на консоль:

pos = 3

pos = 10

pos = 14

**int** max\_size()

*Аргументи: відсутні.*

*Значення, що повертається: максимально можлива довжина рядка.*

string replace(**int** start, **int** num, string str)

*Аргументи:*

*start – номер початкового символа послідовності, що буде замінено;*

*num – кількість символів рядка, що буде замінено;*

*str – послідовність символів, яка буде розміщена у рядку замість замінених.*

*Значення, що повертається: новий рядок.*

**void** resize(**int** size)

*Аргументи:*  
*size – новий розмір рядка.*

*Значення, що повертається: відсутнє.*

**int** rfind(string str, **int** start=-1)

*Аргументи:*

*str – символ, чи послідовність символів для пошуку з кінця рядка;*

*start – номер символа, з якого почнеться пошук послідовності.*

*Значення, що повертається: номер першого символа послідовності, яка шукається з кінця рядка.*

**int** size()

*Аргументи: відсутні.*

*Значення, що повертається: розмір рядка у символах.*

string substr(**int** start, **int** num)

*Аргументи:*

*start – номер початкового символа послідовності;*

*num – кількість символів послідовності.*

*Значення, що повертається: вказана послідовність з рядку.*

**void** swap(string s)

*Аргументи:*

*s – об'єкт, з яким буде виконано обмін рядками.*

*Значення, що повертається: відсутнє.*

Застосування класу роботи з рядками є досить поширеним. Це пов'язано з тим, що більшість інформації, яка обробляється людиною, є текстовою. Окрім того, текстовий формат подання даних є найбільш універсальним.

**Висновки**

У даному розділи було розглянуто наступне питання: клас роботи з рядками та його методи.

**Контрольні питання**

1) Який заголовний файл дозволяє застосовувати клас роботи з рядками?

2) Які прості операції можна виконувати з об'єктами класу роботи з рядками?

3) Які методи класу роботи з рядками дозволяють виконувати пошук символів чи послідовностей символів у рядку?

4) Які методи класу роботи з рядками дозволяють отримати розмір рядка у байтах та символах?

5) Які методи класу роботи з рядками дозволяють працювати з підрядками?

6) Як виконати обмін значеннями між двома об'єктами класу роботи з рядками?

7) За допомогою якого методу можна перевірити чи порожній рядок?

8) За допомогою якого методу можна отримати константний покажчик на символьний масив, що вміщує рядок об'єкта класу роботи з рядками?

**Лекція 19. Робота з файлами**

Навчальною метою розділу є ознайомлення читача про принципи роботи з даними, що зберігаються у вигляді файлів.

Внаслідок вивчення матеріалу даного розділу читач повинен вміти:

– застосовувати в програмі функції роботи з файлами;

– виконувати відкриття файлів у різних режимах;

– читати та записувати дані;

– працювати з курсором файлу;

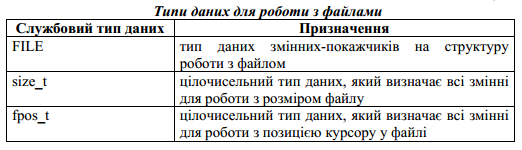
– закривати файл.

**19.1. Файловий ввід/вивід за допомогою потокових функцій**

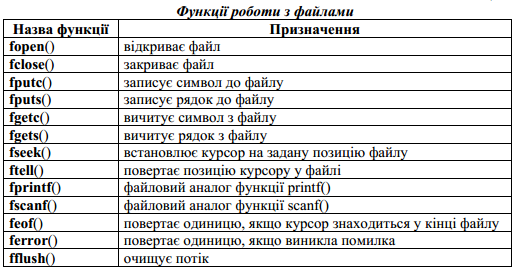
Файл – це дані, що збережено на носії інформації. Для роботи з файлами у мові С++ найчастіше застосовується ввід/вивід на основі потокових функцій.

Всі необхідні функції, макроси та типи даних, що забезпечують повний доступ до файлів, входять до вже знайомої бібліотеки "stdio.h". Для оперування файловою інформацією достатньо знати наступні функції, та типи даних, наведені у таблицях 19.1 й 19.2.

Таблиця 19.1

******

Таблиця 19.2

******

Формати оголошень функцій, що показані у таблиці, та приклади їх застосування наведено нижче:

FILE \*fopen(**const char** \**file\_name*, **const char** \**mode*);

*Аргументи:*  
*file\_name – шлях до файлу;*

*mode – режим відкриття файлу.*

*Значення, що повертається: адресу структури роботи з файлом, яка буде застосована для всіх функціях.*

*Можливо використовувати наступні режими:*

*r – відкрити текстовий файл для читання;*

*w – створити текстовий файл для запису;*

*a – додати у кінець текстового файлу;*

*rb – відкрити бінарний файл для читання;*

*wb – створити бінарний файл для запису;*

*ab – додати у кінець бінарного файлу;*

*r+ – відкрити текстовий файл для читання/запису;*

*w+ – створити текстовий файл для читання/запису;*

*a+ – додати у кінець текстового файлу чи створити текстовий файл для читання/запису;*

*r+b – відкрити бінарний файл для читання/запису;*

*w+b – створити бінарний файл для читання/запису;*

*a+b – додати у кінець бінарного файлу чи створити бінарний файл для читання/запису.*

**int** fclose(FILE \**fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом.*

*Значення, що повертається: 1 – якщо виникла помилка.*

**int** fputc(**int** *c*, FILE \**fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом;*

*с – символ для запису.*

*Значення, що повертається: позиція курсору.*

**int** fputs(**const char** \**string*, FILE \**fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом;*

*string – константний рядок для запису.*

*Значення, що повертається: позиція курсору.*

**int** fgetc(FILE \**fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом.*

*Значення, що повертається: символ.*

**char** \*fgets(**char** \**string*, **int** *n*, FILE \**fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом;*

*string – адреса символьного рядка;*

*n – кількість символів.*

*Значення, що повертається: адреса символьного рядка.*

**int** fseek( FILE \**fp*, **long** *offset*, **int** *origin* );

*Аргументи:*

*fp – адреса структури роботи з файлом;*

*offset – крок зсуву;*

*origin – макрос визначення початковоїпозиціїдля зсуву курсору.*

*Можливо використовувати наступні макроси:*

*SEEK\_CUR – починати з позиції курсору;*

*SEEK\_END – починати з кінця файлу;*

*SEEK\_SET – починати з початку файлу.*

*Значення, що повертається: нуль, якщо була виповнена успішно.*

**long** ftell( FILE \* *fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом.*

*Значення, що повертається: позиція курсору у файлі.*

**int** fprintf( FILE \* *fp*, **const char** \**format* [, *argument* ]...);

*Аргументи:*

*fp – адреса структури роботи з файлом;*

*format – рядок, що форматується;*

*argument – список змінних.*

*Значення, що повертається: кількість записаних байтів.*

**int** fscanf( FILE \* *fp*, **const char** \**format* [, *argument* ]... );

*Аргументи:*

*fp – адреса структури роботи з файлом;*

*format – рядок, що форматується;*

*argument – список змінних.*

*Значення, що повертається: кількість зчитаних байтів.*

**int** feof( FILE \* *fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом.*

*Значення, що повертається: значення EOF, якщо курсор у кінці файлу.*

**int** ferror( FILE \* *fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом.*

*Значення, що повертається: номер помилки чи нуль, якщо помилки не було.*

**int** fflush( FILE \* *fp*);

*Аргументи:*

*fp – адреса структури роботи з файлом.*

*Значення, що повертається: нуль, якщо виконана успішно, та EOF, якщо виникла помилка.*

Приклад програми, що відкриває текстовий файл та виводить його зміст на консоль:

#**include** <stdio.h> //*підключення бібліотеки вводу/виводу*

**void** main() //*оголошення та визначення головноїфункції*

{

//*оголошення змінної-покажчика на структуру роботи з файлом*

FILE \*f;

f=fopen("text.txt", "r"); //*відкриття файлу text.txt для читання*

**while** (!feof(f)) //*цикл до визначення кінця файлу*

{

printf("%c", fgetc(f)); //*посимвольний вивід тексту на консоль*

}

printf("\r\n"); //*перехід на наступний рядок*

fclose(f); //*закриття файлу*

}

Приклад програми, що записує рядок "This file was created by \_" з введеним користувачем ім'ям у файл output.txt.

#**include** <stdio.h> //*підключення бібліотеки вводу/виводу*

#**include** <string> //*підключення бібліотеки роботи з рядком*

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

**using namespace** std; //*застосування простору імен*

**void** main() //*оголошення та визначення головноїфункції*

{

//*оголошення змінної-покажчика на структуру роботи з файлом*

FILE \*f;

string st; //*оголошення об'єкта класу роботи з рядками*

f=fopen("output.txt", "w"); //*створення файлу output.txt для запису*

**cout**<<"Your name: "; //*вивід запиту до користувача*

**cin**>>st; //*отримання значення від користувача*

fprintf(f, "This file was created by %s", st.c\_str()); //*запис тексту у файл*

fclose(f); //*закриття файлу*

}

Приклад програми, що вичитує текст з файлу text.txt та формує з нього у файлі output.txt псевдографічний трикутник:

#**include** <stdio.h> //*підключення бібліотеки вводу/виводу*

#**include** <string> //*підключення бібліотеки роботи з рядком*

#**include** <iostream> //*підключення бібліотеки потокового вводу/виводу*

**using namespace** std; //*застосування простору імен*

**void** main() //*оголошення та визначення головноїфункції*

{

//*оголошення змінної-покажчика на структуру роботи з файлом*

FILE \*f;

string st, sn; //*оголошення об'єктів класу роботи з рядками*

**int** n=0;//*оголошення та визначення змінноїлічильника номера символа*

f=fopen("text.txt", "r"); //*відкриття файлу text.txt для читання*

**while** (!feof(f)) //*цикл до визначення кінця рядка*

{

st+=fgetc(f); //*формуємо рядок*

}

fclose(f); //*закриття файлу*

**for** (**int** i=0; i<15; i++) //*цикл за рядками трикутника*

{

**for** (int j=0; j<(15-i); j++) //*цикл для додавання відступів*

sn+=' '; //*додати у рядок символ пробілу*

**for** (**int** k=0; k<2\*i+1; k++)//*цикл для додавання символів тексту*

sn+=st.at(n); //*додати один символ у рядок*

n++; //*збільшення номера символа на одиницю*

}

sn+="\r\n"; //*додати перехід на новий рядок*

}

**cout**<<sn; //*вивід результату на консоль*

f=fopen("output.txt", "w"); //*створення файлу для запису*

fprintf(f, "%s", sn.c\_str()); //*запис рядка у файл*

fclose(f); //*закриття файлу*

}

**Висновки**

У даному розділі було розглянуто наступне питання: функції для потокової роботи з даними, що збережено у вигляді файлів.

**Контрольні питання**

1) Який заголовний файл підключає функції роботи з файлами?

2) Які спеціальні службові типи даних зазначено для роботи з файлами?

3) За допомогою якої функції виконується відкриття файлу?

4) У яких режимах може бути відкрито файл?

5) Яка функція дозволяє отримати позицію курсору файлу?

6) За допомогою яких функцій виконується читання тексту з файлу?

7) Як у програмному коді перевірити чи досягнуто при читанні кінець файлу?

8) Які значення містять змінні типу FILE?

9) Які функції виконують запис даних у файл?

**Лекція 20. Візуальне програмування**

**20.1. Основні поняття.**

Технологія роботи у середовищі С++ Builder базується на ідеях об'єктно-орієнтованого та візуального програмування. Ідея об'єктно-орієнтованого програмування полягає в інкапсуляції (об'єднанні) даних і засобів їх опрацювання (методів) у тип, який називається класом. Конкретною змінною певного класу і є об'єкт. Прикладами об'єктів можуть бути елементи керування у вікні: кнопки, списки, текстові поля тощо. Середовище візуального програмування С++ Builder - це графічна автоматизована оболонка над об'єктно-орієнтованою мовою програмування С++ . Якщо у мові Сі структурними одиницями є дані та команди, то тут такою структурною одиницею є візуальний об'єкт, який називається кo;rtnoueumoм. Автоматизація програмування досягається завдяки можливості переносити компонент на форму (у програму) з палітри компонентів і змінювати його властивості, не вносячи вручну змін до програмного коду.

Формою називають компонент, який володіє властивостями вікна Windows і призначений для розташування інших компонентів. Компоненти на формі можуть бути видимими та невидимими. Видмі призначені для організації діалогу з користувачем. Це різні кнопки, списки, текстові поля, зображення тощо. Вони відображаються на екрані під час виконання програми. Невидимі компоненти призначені для доступу до системних ресурсів комп'ютера.

Проект - це сукупність файлів, з яких складає'гься С++  
Builder-пpoгpaмa.

**20. 2. Інструменти середовища С++ Builder.**

Вікно середовища містить головне меню, панелі інструментів (ToolBars), а також:

• палітру компонентів (Component Palette);

• вікно властивостей об'єктів (Object Inspector);

• вікно форми;

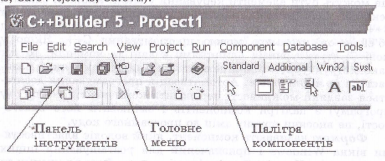
• редактор коду програми .

Ці інструменти стають доступними після запуску С++ Builder: три знаходяться у головному вікні (верхня частина екрана, рис. 20.1), арешта-в окремих вікнах . Ці вікна також можна відкрити командами головного меню View ~ TooiBars ~Component Palette, View ~ Object lnspector, View ~ Forms та View ~ Units. Почергова активізація вікна форми та коду програми здійснюється командою головного меню View ~ Toggle Form/Unit чи клавішею F12.

**20.3. Головне меню та панель інструментів**.

Головне меню складається з таких елементів: File, Edit, Search, View, Project, Run, Component, Database, Tools, НеІр (див. рис.20.1).

Меню File містить стандартні команди для роботи з файлами проекту. За допомогою цих команд можна створити новий проект (New Application), нову форму (New Form), відкрити чи закрити файл проекту (Open і Close), закрити всі відкриті файли (Ciose АІІ), зберегти файл, проект або все відразу (Save, Save As, Save Project As).



***Рис. 20.1. Головне вікно С++ Builder***

За допомогою команд меню Edit можна вирівнювати компоненти відносно сітки та між собою (Aiign to Grid, Align), задавати порядок відображення компонентів, які перетинаютьс (Bring to Front, Send to Back), змінювати розмір вибраного компонента (Size~, масштабувати візуальні компоненти (Scale) тощо.

Меню Search містить стандартні команди пошуку та заміни фрагмента тексту (Find, Replace, Search Again, lncremental Search) та інші. У меню View знаходяться команди візуалізації елементів середовища. Меню Project містить команди керування проектом, зокрема команди додавання файлів до проекту (Add to Project) та команди компіляції (Compile Unit, Build АІІ Project). Меню Run містить команди налагодження та запуску програми. Меню Component використовують для створення та інсталяції нових компонентів. Меню Database містить команди виклику інструментів бази даних. У меню Tools містяться команди для задання параметрів середовища.

Панель інструментів служить для розташування кнопок інструментів. На ній можуть міститися кнопки всіх зазначених команд.

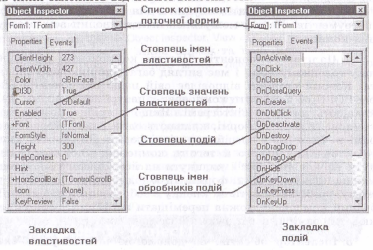
**20. 4. Палітра компонентів.**

Палітра компонентів розташована у головному вікні і має вигЛяд багатосторінкового блокнота. Кожній сторінці відповідає свій набір компонентів (див.рис. 20.1). Щоб помістити компонент у центрі вікна форми, двічі клацають на його піктограмі. Якщо потрібно розташувати компонент десь на формі, клацають один раз на його піктограмі та один раз у потрібному місці форми. Для багаторазового вставляння одного й того ж компонента потрібно натиснути на клавішу Shift і клацнути на його піктограмі – тепер можна клацати у вікні форми. Щоб відмовитися від цього режиму, треба натиснути на кнопку з зображенням стрілки. Вибраний компонент можна переміщати на формі, а також змінювати розміри, перетягуючи його маркери.

**20.5. Інспектор об'єктів**.

За допомогою інспектора об'єктів можна задавати початкові значення властивостей об'єкта та їхню реакцію на стандартні події. Вікно інспектора об'єктів містить список компонентів поточної форми, а також дві закладки: властивостей (Properties) та подій (Events). Щоб активізувати вікно інспектора об'єктів, використовують клавішу F11. Розглянемо це вікно (рис. 20.2). Закладка властивостей складається з двох стовпців: лівий містить назви властивостей компонентів, а правий - їхні значення. Властивості можуть бути простими або комплексними. Комплексні властивості складаються з набору інших властивостей. Такі властивості в іспекторі об'єктів позначені символом "+", наприклад +Font. Закладка подій також має два стовпці. У лівому відображаються імена стандартних подій, на які об'єкт може реагувати, а в правому - імена методів-обробників (функцій), які реалізовуватимуть реакцію на подію. Кожній стандартній події відповідає назва методу, яка з' являється після подвійного клацання мишею у правому стовпці. У цей момент у вікно тексту програми додається шаблон базового коду (функції) для відповідного методу, який треба заповнити.

Для введення значень властивостей числового і текстового типу (Width, Name тощо) використовується стандартне поле введення. Значення властивостей перерахованого типу (Aiign, Cursor тощо) задаються комбінованим списком, звідки вибирають потрібне. Деякі комплексні властивості (Font, Picture, Glyph тощо) використовують діалогові вікна, набір керуючих елементів яких залежить від певної властивості.



***Рис. 20.2. Вікно Інспектора об' єктів***

**6. Вікно форми.**

Форма - це вікно Windows, яке утворюється в одному з можливих для вікон стилів. Увесь внутрішній простір є робочою областю, яка має сітку вирівнювання для зручного розташування компонентів на формі. Для виконання групових операцій декілька компонентів можна об'єднуватИ. Для цього необхідно натиснути на ліву клавішу миші і переміщенням вказівника охопити всі потрібні компоненти.

У групу долучаються компоненти, які хоча б частково потрап ляють в охоплену область. Можна також долучити/вилучити окремий елемент. Для цього необхідно натиснути на клавішу Shift та, не відпускаючи її, вибрати мишею потрібний компонент на формі. Вилучення виокремлених компонентів чи групи виконується клавішею Delete, переміщення виокремленого компонента в межах форми мишею. Над компонентами та їхніми групами можна виконувати операції вирізання, копіювання в буфер обміну та вставляння з буфера.

Вирівнювати компоненти можна як відносно вікна форми, так і один відносно одного. Для цього використовується команда Edit => Align головного меню або палітра вирівнювання (команда View => Alignment Palette головного меню). Інша можливість - безпосередньо задати властивості Left і Тор компонентів. Компоненти у групі вирівнюються відносно того компонента, який потрапив у групу першим.

**7. Структура проекту**.

Проектом називають сукупність файлів, з яких С++ Builder створює готову для виконання програму. До складу кожного проекту обов' язково входять такі файли:

файл проекту \*.bpr. Це невеликий файл, який містить посилання на всі файли проекту й ініціалізує програму;

• файли опису всіх форм, які входять у проект: файл тексту програми \*.срр файл форми \*.dfm;

файл ресурсів програми \*.res. У ньому описані ресурси, які не входять у форму, наприклад, піктограма програми;

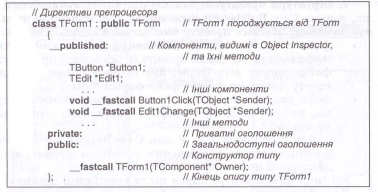
• файли заголовків ''<.h. У них С++ Builder автоматично розміщує описи породжених класів, а користувач, декларації новостворених функцій;

У сі інші файли створюються після компіляції проекту.

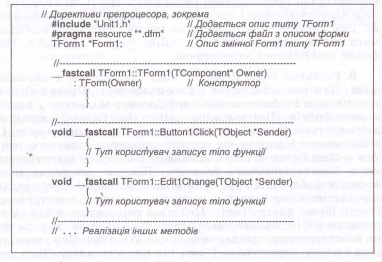
Для збереження проекту необхідно задати імена модулів (автоматично пропонуються імена Unitl.cpp, Unit2.cpp, ... ) та ім'я проекту (Projectl.bpr). Ці імена можна змінити на власні. Для переміщення проекту на інший комп'ютер необхідно мати файли таких типів: \*.bpr, \*.dfm, \*.срр, \*.res та '<.h. Інші файли створюються автоматично.

**8.Редактор коду.**

Редактор коду знаходиться в окремому вікні. Це вікно складається з двох половинок. Права половина організована як багатосторінкавий блокнот відкритих у певний момент файлів. Ліва половина- вікно Class Explorer, у якому відображається ієрархія створених класів і функції проекту. Це вікно можна відкрити чи закрити командою головного меню View ~ Class Explorer а також командою View Explorer контекстового меню багатосторінкового блокнота. Під час створення нової форми у файл Unitl.cpp, який відповідає формі Form1, редактор автоматично заносить порожню заготовку конструктора TForm1 ::TForm1 класу TForm1. Цей клас пораджується від інтегрованого уС++ Builder класу TForm (Форма), який володіє усіма властивостями стандартного вікна Windows. Опис наслідування класу користувача TForm1 від готового класу TForm знаходиться у загаловочному файлі Unitl.h. Посилання на цей файл **(include),** а також опис змінної Form1 типу TForm1 розміщені у модулі Umitl.cpp Під час додавання нових компонентів до форми у файл Unit.dfm автоматично заносяться описи параметрів цих компонентів (висота, ширина, розташування, стиль тощо). Застосування методу до певного об'єкта веде до появи заготовки базового коду відповідної функції у вікні редактора. Заготовка (шаблон) складається із заголовка функції та операторних дужок { }. Заготовку заповнює користувач. Файл Unitl.h має такий вигляд:



Файл Unitl.cpp має такий загальний вигляд:



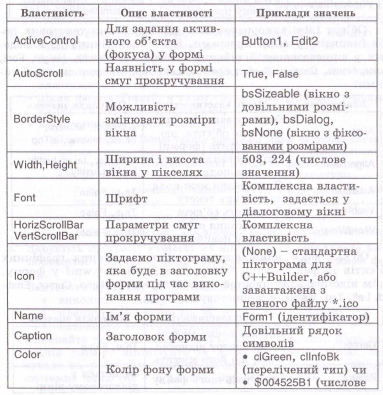
**Лекція 21. Задача про анкету та обмін валюти**

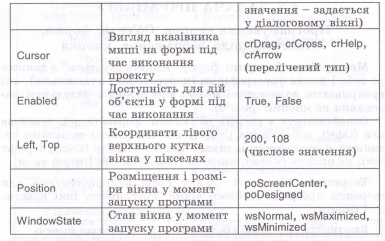
**21.1.3адача про анкету**

**Програмування кнопок. Об'єкти: форма, текстове поле, зображення, кнопка**

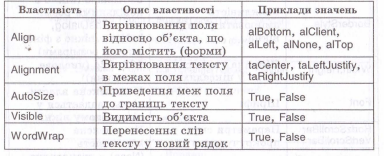
**Мета роботи**. Створити форму "Анкета студента" з даними про себе і двома фотографіями '(портретною і художньою), які перекривають одна одну і мають з'являтися в результаті натискання на кнопки (рис . 21.1). Ознайомитися з такими об'єктами: форма (Form), текстове поле (Label), зображення (lmage), кнопка (Button) та їхніми основними властивостями: підпис (Caption), колір (Color), шрифт (Font), видимість (Visible), ширина (Width), висота (Height) та ін.

Теоретичні відомості. Об'єкт Form використовують для створення програмою нового вікна. Розглянемо такі властивості форми:

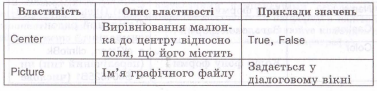


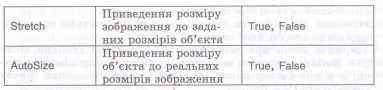


Об'єкт Label використовують для створення текстових полів (написів) у вікні програми. Крім аналогічних до наведених у вищезазначеній таблиці властивостей Width, Height, Font,Color, Name, Caption, Cursor, Enabled, Left, Тор, він володіє ще й такими:



Об'єкт ' lmage використовують для вставляння графічних об'єктів із файлів типу \*.bmp, \*.emf, \*.ісо, \*.wmf у форму. Крім відомих властивостей Align, Width, Height, Name, Cursor, Enabled, Left, Тор, Visible, використовують такі:





Об'єкт Button використовують для створення кнопок на формі. Кцопки мають такі властивості: Visible, Width, Height, Font, Color, Name, Caption, Cursor, Enabled, Left, Тор та ін.

**Хід роботи**

1. Завантажте середовище візуального програмування Borland С++ Builder. Запуск системи візуального програмування С++ Builder виконують клацанням на піктограмі fl.il або за допомогою каскадного меню Start (Пуск) =} Programs (Програми) =} Borland С++ Builder х.О =}С++ Builder х.О, де х- версія програми. Отримаємо чотири вікна.

2. Дослідіть способи активізації чотирьох вікон С++ Builder:

• головного вікна С++ Builder х.О - Projectl, де є панель інструментів, палітра компонентів і головне меню;

• вікна інспектора об'єктів Object Inspector зі значеннями властивостей активного об'єкта;

• вікна форми Forml, в якому будуть розташовані результати роботи майбутньої програми;

• вікна тексту програми (Unitl.cpp).

Зауважепн.я 1. Вікно тексту програми може частково перекриватися вікном форми. Активізувати вікна (а також змінювати їхні розміри чи розташування) можна за допомогою миші або використовуючи функціональні клавіші на клавіатурі:

F10 -для активізації головного меню (після цього натисніть на клавішу Esc);

F11 - для активізації вікна інспектора об'єкта;

F12 - для переходу між вікнами форми та коду програми.

3. Запустіть програму Projectl на виконання і розгляньте вікно порожньої поки що форми. Поекспериментуйте з вікном форми.

Запустити програму можна декількома способами:

• виконати команду Run =} Run головного меню;

• клацнути на кнопці Run І> \ панелі інструментів;

• натиснути на функціональну клавішу F9.

Виконайте такі дії: максимізуйте вікно, відновіть його попередюи розмір, мшlмшуйте та знову розгорніть вікно, пересуньте на робочому столі та змініть його розміри, викличтесистемне меню (Ait + пропуск). Виконайте ті самі дії за допомогою команд Move, Size та інших і клавіатури.

Висновок: вікно форми володіє усіма властивостями стандартного вікна операційної системи Windows.

4. Закрийте вікно програми Forml, мінімізуйте головне вікно С++ Builder і створіть на робочому диску папку з іменем групи, а у ній власну папку, названу вашим прізвищем. Знову активізуйте С++ Builder.

5. Збіfежіть створену програму у своїй папці. Для цього виберіть команду головного меню File => Save дІІ (Зберегти Все) або натисніть на кнопку Save all на панелі інструментів. У першому рядку вікна, яке з'явиться ("Save Unitl As") під заголовком "Save in:" (Зберегти в:), за допомогою випадаючого меню .::.1 виберіть ім' я робочого диска, після чогознайдіть і відкрийте свою власну папку . Задайте назву дляфайлу тексту програми, попередньо витерши запропоновану комп'ютером назву Unitl.cpp, => Save. У наступному вікні "Save Projectl As" дайте назву файлові проекту, витерши запропоновану комп'ютером назву Projectl.bpr => Save. Зверніть увагу:файли проекту і тексту програми повинні мати різні назви.

6. Візуально ознайомтеся з властивостями форми Left, Тор, Width та Height. Перемістіть за допомогою миші форму Forml. Зверніть увагу, що зміна розташування форми веде до зміни її властивостей Left та Тор - координат лівого верхнього кута форми у вікні Object Inspector. Змініть розміри форми. Переконайтесь, що тепер змінюються властивості Width (ширина) та Height (висота) форми у вікні інспектора об'єкта.

7. Дослідіть, як зміна значень властивостей Left, Тор, Width чи Height форми у вікні Object Inspector призводить до зміни розташування чи розміру форми. Еведіть відповідне значення у пікеелях і натисніть на клавішу Enter.

8. Змініть колір фону форми . Для цього у вікні властивостей форми Object Inspector у рядку Color виберіть значення кольору фону двома способами:

• викличте вікно вибору кольору подвійним клацанням мишею на поточному значенні властивості СоІог. Виберіть один із базових кольорів (Basic colors) або встановіть свій власний (Define Custom Colors) кqлір. Підтвердіть вибір (Ok). ·

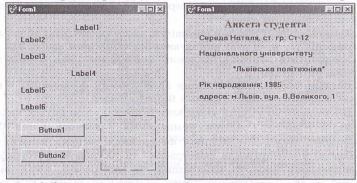
• за допомогою випадаючого меню ~ поекспериментуйте з різними значеннями властивості Color. Задайте початкове значення кольору ciBtnFace.

9 . Виконайте програму ще раз (див. п. 3).

10. Вставте у форму текстове поле (об'єкт типу Label) з текстом "Анкета студента" .Двічі клацніть мишею на піктограмі Label на закладці Standard палітри компонентів головного вікна C++Builder. Розташуйте вставлений об'єкт, наприклад, так, як показано на рис. 21.1, переміщуючи його мишею. Якщо об'єкт Label1 невиокремлений, активізуйте його і у вікні Object Inspector змініт значення властивості Caption з Label1 на текст "Анкета студента" без лапок. Змініть знач~ння властивості Font (шрифт) цього текстового поля на такі: Font : Times New Roman Cyr;• Font style : Bold;Size : 16;Color : Purple.

Зауваження 2. У вікні Object Inspector відображається списоквластивостей лише активного у певний момент об'єкта.

11. Аналогічно вставте у форму ще декілька текстових полів звашими біографічними даними.Один із варіантів розміщення текстових полів показаний на рис. 21.2

.

**Рис. 21.1. Створення форми Рис . 21.2. Приклад підписів**

12. Вставте у форму об'єкт тиnу lmage (зображення). Для цього клацніть один раз лівою клавішею миші на піктограмі lmage ~ закладки Additional (додаткові) палітри компонентів і, наприклад, у нижньому правому куті форми обведіть контур для майбутнього зображення (фотографії). Якщо потрібно, змініть розмір форми чи вставленого об'єкта та добийтеся якнайкращого розташування на ній створених раніше об'єктів. Змінювати розміри об'єкта можна методом їх "розтягування" за маркери (чорні габаритні квадратики). Запам'ятайте назву, яку Builder присвоїть цьому об'єкту (значення властивості Name) або замініть її на свій розсуд. За замовчуванням цей об'єкт матиме стандартну назву lmage1.

13. Вставте свою портретну фотографію за доnомогою властивості Picture (ілюстрація) об'єкта lmage1.Для цього виокремте об'єкт lmage1 та активізуйте рядок Picture у вікні Object Inspector. Клацнувши на кнопці ,•::, викличте діалогове , вікно вибору малюнка Picture Editor. Клацніть на кнопці Load (завантажити) і у вікні Load picture зазначте шлях до файлу з фотографією. Якщо такого файлу немає, скористайтесь будь-якою фотографією з бібліотеки 16color, яка за замовчуванням знаходиться у папці С:\ Program Files \ Common Files \ Borland Shared \ Images \ Splash \ 16color. Виберіть будь який файл => Open. Підтвердіть свій вибір у вікні Picture Еdіtо tі- натисканням на клавішу Ok. Задайте властвість Stretch дляоб єкта lmage1 як True.

14. Вставте свою художню фотографію у форму поверх наявної, скориставшись ще одним об'єктом типу lmage. Один із варіантів розташування фотографії показаний на рис. 21.4. Вважатимемо, що цей об'єкт має назву lmage2.

Зауваження 3. Під час накладання об'єктів може виникнути потреба використати команди Send То Back (переслати назад) чи Bring То Front (перенести наперед), які є в їх контекстових меню чи в головному меню Edit.

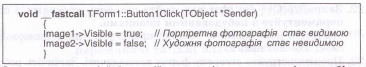
15. Поекспериментуйте з властивістю Visible (видимість) обох зображень, кожного разу виконуючи програму (див. пункт 3). Після цього виберіть значення властивості Visible у False для обох зображень.

16. Вставте у форму кнопки для засвічування фотографій – два об'єкти типу Button з назвами Button1 і Button2. Піктограма ОО об'єкта типу Button (кнопка) знаходиться на закладці Standard палітри компонентів головного вікна С++ Builder. Поміняйте підписи на кнопках (змініть властивість Caption) на "Портретна фотографія " та "Художня фотографія" відповідно. Виберіть найкращий, на ваш розсуд, кирилізований шрифт для .підписів. Якщо використано картинки із стандартної бібліотеки Borland, виберіть для кнопок цікаві підписи. Один із варіантів розміщення кнопок показаний на рис. 21.5.

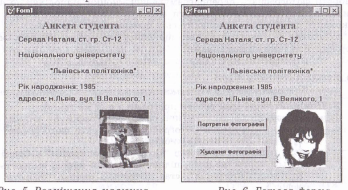
17. Запрограмуйте кнопку "Портретна фотографія" так, щоб після ії натискання у формі з'являлась портретва фотографія. Button1 необхідно двічі клацнути на ній лівою клавішею миші. У результаті активізується вікно тексту програми із заготовкою функції Button1Ciick, яка опрацьовуватиме подію клацання на кнопці Button1:



У заготовку необхідно вставити текст програми реакції на цю под'ію. Процедура матиме такий вигляд:

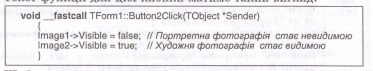


За допомогою цієї функції властивість видимості для об'єкта Іmage1 задаємо і цю ж властивість для об'єкта lmage2 забираємо. Для кнопки "Художня . фотографія" дії будуть протилежні. Зверніть увагу на використання складених імен, наприклад lmage1.Visible, в яких назва об'єкта від його властивості відокремлюється крапкою. Такі складені імена дають доступ до значення конкретної властивості деякого об'єкта.



**Рис. 21. 4. Розміщення малюнка Рис. 21. 5. Готова форма**

18. Запрограмуйте кнопку "Художня фотографія" відповідно до їі призначення (див. п. 17). Текст функції для цієї кнопки матиме такий вигляд:



Щоб створити таку функцію швидко, можна скопіювати дві команди присвоєння з попередньої функції у нову і поміняти вирази праворуч.

19. Виконайте програму і переконайтесь, що кнопки виконують свої функції. Закрийте вікно програми "Анкета студента".

20. Збережіть створену програму у своїй папці. Виберіть елемент головного меню File ~ Save all (Зберегти все) або натисніть на кнопку Save all ~на панелі інструментів.

21. Створіть ехе-фай,л програми. Виконайте команду головного меню Project ~ Make Project1(Сконструювати все).

22. Закрийте С++ Builder, виконайте створену програму і поекспериментуйте з побудованими кнопками. ,Запустіть ехе-файл з іменем проекту і піктограмою В зісвоєї власної папки.

23. Продемонструйте створену форму викладачеві. Закінчіть роботу.

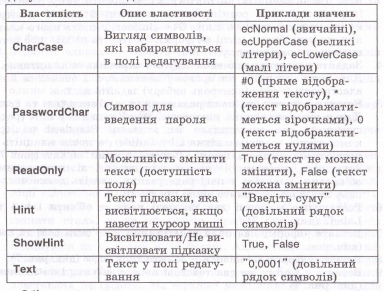
**21.2 3адача про обмін валюти**

**Програмування розгалужень. Об'єкти: поля редагування, перемикачі та їхні властивості**

**Мета роботи.** Створити форму з назвою "Обмін валюти", на якій можна змоделювати операції обміну валюти в обмінному пункті. Застосувати полЯ редагування (Edit) та перемикачі (RadioButton, дослівно радіокнопка), а також кнопки для виконання · обчислень і закінчення роботи програми (див. зразок форми на рис. 21.6).

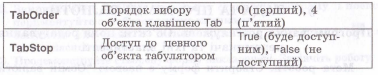
**Теоретичні відомості.**

Об'єкт Edit використовують для введення користувачем рядка символів із клавіатури. У разі необхідності для перетворення одержаного рядка (властивість Text) у число і навпаки використовують функції С++ Builder StrToFioat та FloatТoStr. Окрім відомих вам властивостей, поля редагування Edit володіють такими:



Об'єкти RadioButton використовують для створення у формі засобу для вибирання однієї альтернативної можливості серед декількох. Розглянемо такі властивості перемикачів:





**Хід роботи**

1. Завантажте середовшце візуального програмування С++ Builder.

2. Відмовтесь від можливості змінювати розміри вікна nрограми, надавши властивості форми BorderStyle значення bsDialog. Задавши це значення, виконайте програму і nереконайтеся, що не можна змінити розмір форми. Зверніть увагу на відсутність у вікні кнопок мінімізації і максимізації, а також системного меню. Завершіть роботу програми.

3. Вставте у форму два об'єкти тиnу RadioButton (nеремикачі) як це nоказано на рис. 21.7. Для цього клацніть на піктограмі ~ об'єкта тиnу RadioButton (nеремикач), яка розміщена на закладці Standard палітри компонентів головного вікна С++ Builder, а після цього клацніть на потрібному місці на формі. Повторіть ці дії, щоб вставити другу радіокноnку.

4. Задайте nочаткове значення другого nеремикача як активне. Для цього клацніть на правому перемикачі і значення його властивості Checked (контроль вибору) задайте як True .

5. Вставте у форму два nоля редагування- об'єкти Edit1 та Edit2. Для цього клацніть на піктограмі •1illl об'єкта типу Edit (редагування), яка знаходиться на закладці Standard палітри компонентів головного вікна С++ Builder, а потім клацніть у потрібному місці на формі. Вставте другий об'єкт (рис. 21.7). Запустіть програму поекспериментуйте зі вставленими об'єктами: кл~цніть у полі редагування, введіть деяке число, вилучіть його. Закрийте вікно програми.

6. Розташуйте у формі два текстові nоля - об'єкти Label1 та Label2 (див. рис. 21.7).

7. Вставте у форму два nоля редагування- об'єкти EditЗ та Edit4 (див. pltc. 7).

8. Вставте у форму дві кнопки - об'єкти тиnу Button (див. рис. 7).

9. Вставте у форму ще два текстові поля -об'єкти LаЬеІЗ і Label4 (див. рис. 7).

10. Збережіть створену у цей момент форму у своїй паnці. File ==} Save дІІ. Файли тексту програми та проекту назвіть різними іменами. Імена занотуйте у звіт. У подальшому періодично, зокрема перед черговими запусками проекту на виконання, зберігайте файли програми (File ==} Save all, вводити імена файлів вже не потрібно).



**Рис. 21.7. Створення форми Рис. 21.8. Готова форма**

11. Змініть назву форми з "Form1" на "Обмін валюти". Для цього змініть значення властивості Caption форми. Клацніть на формі і на рядку Caption у вікні Object Inspector. Введітьназву форми без лапок. Зверніть увагу на те, що для об'єктів багатьох типів (зокрема, Label, Button, Form, RadioButton, CheckBox та інших) значення властивостей Caption та Name збігаються.

12. Змініть підписи Caption на об'єктах типу RadioButton, Label та Button так, як показано на рис. 21.8. Для цього по черзі вибирайте об'єкти (клацайте на них) і змінюйте значення властивості Caption.

13. Задайте однакові розміри для всіх текстових полів, полів редагування та кнопок і вирівняйте їх на формі. Для цього одночасно виокремте п'ять об'єктів лівого стовпця одним із способів:

• тримаючи натиснутою клавішу Shift, почергово активізуйте об'єкти, клацаючи на них лівою клавішею миші;

• обведіть навколо цих об'єктів контур, утримуючи натиснутою ліву клавішу миші.

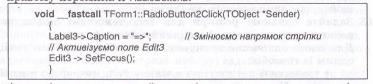
У вікні Object Inspector задайте спільні для цих об'єктів значення властивостей Width (ширина), Height (висота) та Left (відстул від лівої межі вікна) у пікселях. Зверпіть увагу, що пjсля активізації відповідпаї властивості ії зпачепия - це зпачеппя властивості першого виокремлеиого об' єкта створепаї групи. Можете змінити стиль, колір чи розмір шрифта одночасно для усіх виокремлених об'єктів (властивість Font). Зніміть виокремлення, клацнувши на вільному місці форми. Аналогічно виконайте вирівнювання правого стовпця об'єктів. Вирівняйте вставлені поля попарно у горизонтальному напрямку . Для цього змініть властивість Тор (відступ від верхньої межі вікна у пікселях) для відповідних груп об'єктів. Збережіть роботу (Save АІІ).

14. Задайте значення курсів купівлі та продажу валюти, а також кількість валюти, яку кантор купує чи продає. Для цього введіть потрібне число, наприклад 5, як значення властивості Text об'єкта Edit1. Повторіть це для об'єкта Edit2 (значення 5.2) та EditЗ (значення 20). Для набору символа десяткової крапки використайте символ, передбачений операційною системою комп'ютера .

15. Очистіть поле редагування Edit4. Для цього вилучіть значення властивості Text для об'єкта Edit4. Не сплутайте значення властивостей Name та Text цих об'єктів.

16. Заблокуйте можливість введення даних для поля Edit4, задавти його властивість ReadOnly як True, оскількИ це поле міститиме результат. 3мінювати значення певної властивості можна подвійним клацанням на ній лівою клавішею миші. Збережіть роботу (Save АІІ). Виконайте програму і переконайтеся, що не можна ввести чи редагувати дані у полі Edit4.

17. Запрограмуйте радіокнопки так, щоб напрямок стрілки показував на вид операції: купівля чи продаж. Зробіть активним поле EditЗ. Клацніть двічі на правому перемикачі RadioButton2 (Продаж). ОтримаЄте заготовку функції RadioButton2Ciick. У тілі цієї функ ції опишіть дії, які мають відбутися у результаті клацання на правому перемикачі RadioButton2:



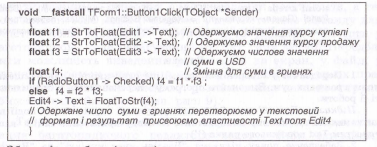
Аналогічно запрограмуйте подію Click клацанням на лівому перемикачі RadioButton1, врахувавши, що стрілка має показувати на ліве поле (' < ='). Фрагмент програмного коду створеної функції запишіт ь у звіт.

18. Запустіть програму і переконайтесь, що перемикач виконує свої функції згідно з п. 17.

19. Запрограмуйте кнопку "Кінець" . Скористайтесь функцією закінчення роботи програми exit():



Дане у полі редагування - це значення властивості Text об'єкта типу рядок. Для перетворення цього даного у числовий дійсний тип (float) скористайтесь функцією StrToFioat(), а навпаки - функцією FloatToStr(). Опишіть основні (kurs, suma) і додаткові (cod, ed) змінні .



11 Одержане число суми в гривнях перетворюємо у текстовий формат і результат присвоюємо властивості Text поля Edit4.

21. Збережіть роботу (Save АН).

22. Виконайте програму і проекспериментуйте з різними грошовими сумами й операціями купівлі чи продажу. Закрийте вікно програми "Обмін валюти". Для переривання роботи програми у випадку неправильного введення вхідних даних виконайте пункт головного меню Run=> Program Reset.

23. Змініть розміри та кольори символів (зокрема об'єктів Lаbel і EditЗ), розташування об'єктів, фон форми (властивість Color) так, щоб форма виглядала якнайкраще.

24. Забезпечте появу підказки "Введіть суму в доларах" після переміщення вказівника миші до поля EditЗ.

Виберіть об'єкт EditЗ і встановіть властивість ShowHint у True, а як значення властивості Hint введіть текст підказки. Збережіть роботу, запустіть програму і переконайтеся, що підказка з'являється.

25. Поміняйте вигляд стрілки з=> нас:::>, а<= на<::::>. Для цього виберіть об'єкт LаЬеІЗ і як значення властивості Caption введіть українську букву р, після чого, активізувавши властивість Font, виберіть назву шрифта Wingdings . Двічі клацніть на правому перемикачі і в його функції введіть українську букву р замість =>. У функції для лівого перемикача символи <= замініть буквою п. Збережіть роботу, запустіть програму і переконайтеся, що стрілка змінила свій вигляд.

26. Створіть ехе-файл вашої програми. Виконайте пункт головного меню Project => Make Project.

27. Закрийте С++ Builder, запустіть створену програму виконайте обчислення для різних початкових даних. Запустіть ехе-файл з іменем проекту і піктограмою W зі своєї папки.

28. Продемонструйте створену форму викладачеві. Закінчіть роботу.

Зауважеnпя 1. Зверніть увагу на використання коми чи крапки у вхідних даних. "У числах, які стосуються курсу валюти, гривневої чи доларової сум для десяткової крапки використайте символ, передбачений операційною системою вашогокомп'ютера (див. Start (Пуск) =} Settings (Налаштовування) =} ControlPanel (Панель керування) =} Regional Settings (Місцеві nараметри) =}закладка Number (Числа), рядок Decimal symbol (Символ десяткової краnки)).

**Лекція 22. Задача про табулювання функції та навчальна послідовність**

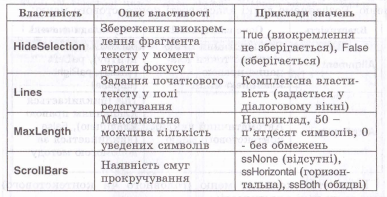
**22.1 Задача табулювання функції**

**Програмування циклів. Об'єкти: Memo, MainMenu,PopupMenu, CheckBox, GroupBox**

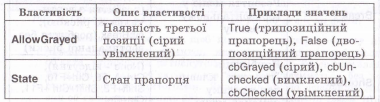
**Мета роботи**. Створити форму для розв' язування задачі табулювання функції. Створити у ній головне меню (об'єкт типу MainMenu) з командами: закінчити роботу програми, табулювати функцію, очистити поля виведення результатів, а також контекстове меню (PopupMenu), що міститиме команду для очистки поля виведення. Результати табулювання вивести у багаторядкове поле редагування (об'єкт типу Memo). Передбачити можливість виведення результатів на екран, у файл, у масив. Напрямок виведення задати за допомогою трьох прапорцін (об'єктів типу CheckBox), ·Розташованих на панелі групи об'єктів (типу. GroupBox) (див. рис. 22.1).

**Теоретичні відомості**.

Об'єкт Memo застосовують для створення багаторядкового редактора тексту. Крім звичайних властивостей, поле редагування Memo володіє ще такими:

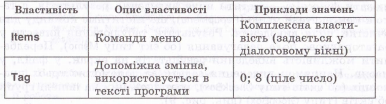


Об'єкт CheckBox використовують для створення незалежного дво- чи трипозиційного прапорця: увімкнено/вимкнено (/недоступний). Для цього об'єкта визначені такі дві нові властивості:

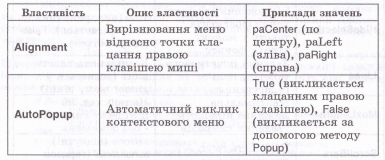


Панель групи об'єктів GroupBox призначена для розміщення на ній групи із кількох об'єктів. Панель групи використовують для покращення дизайну вікна програми. Властивості цього об'єкта аналогічні до описаних вище.

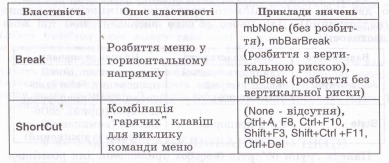
За допомогою об'єкта MainMenu створюють головне меню програми. Ось деякі властивості головного меню:



За допомогою об'єкта PopupMenu створюють контекстове меню деякого компонента. Для "прив'язування" контекстового меню до конкретного об'єкта необхідно його властивості PopupMenu надати значення імені (Name) конкретного контекстового меню. Розглянемо деякі властивості контекстового меню:



Конкретна команда меню (головного чи контекстового) може мати такі властивості:



**. Хід роботи**

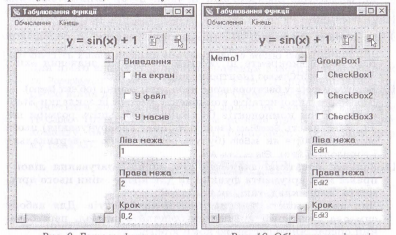
1. Завантажте середовище візуального програмування С++ Builder.

2. Змініть заголовок (Captio~) форми з "Forml" на "Табулювання функції" (без лапок) і збільшіть розміри форми у вертикальному напрямку.

3. Змініть піктограму у лівому верхньому куті форми, задавши конкретний файл з рисунком піктограми як значення властивості Ісоn (піктограма) форми. Клацніть у рядку Ісоn на, а потім на кнопці Load вікна -Picture Editor, щоб отримати вікно Load picture. Відкрийте папку С: \ Program Files \ ,Common Files \ Borland Shared \Images \ Icons, виберіть графічний файл з будь-якою піктограмою ~ Open => Ok.

4. Збережіть виконану на даний момент форму у своїй власній папці (File ~ Save all).

5. Розташуйте у формі поля редагування Edit1, Edit2, EditЗ і відповідні їм текстові поля "Ліва межа", "Права межа", ":Крок", атакож текстове поле для вигляду заданої функції у = sinx +1



**Рис. 22.1. Готова форма Рис. 22.2 О. Об' єкти на формі**

Зауваження 1. Для того, щоб швидко вставити у форму декілька однотипних об'єктів, клацніть на піктограмі цього об'єкта, утримуючи натиснутою клавішу Shift. Тепер вставляпня у форму всіх об'єктів цього типу відбуватиметься без повторного вибору пікnограми. Якщо випадково вставите зайвий об' єкт, то вилучіть його за допомогою клавіші Delete. Щоб відмовитися від такого режиму, клацніть на зображенні стрілки на палітрі компонентів. Розмір, стиль і колір шрифтів виберіть на власний розсуд так, щоб форма виглядала якнайкраще. Save аІІ .

6. Вирівняйте вставлені поля редагування до лівого краю першого об'єкта та відцентруйте текстове поле вигляду функції, скориставшись вікном вирівнювань Alignment. Виокремте групу полів редагування та підписи до них і виконайте команду головного меню Edit (редагувати) => Align (вирівняти) => Horisontal - Left sides (горизонтально - ліві межі) => Vertical - Space equally (вертикально - рівномірно) => Ok. Виокремте текстове поле вигляду функції Edit ~ дlign ~ Horisontal - Center in Window (горизонтально -до центру вікна) => Ok. Save all.

7. Вставте у форму панель групи об'єктів (об'єкт типу GroupBox). Для цього використайте компоненту GroupBox ГJ із закладки Standard. Змініть значення властивості Caption (підпис) цього об'єкта на слово "Виведення" (без лапок). Розмір, стиль і колір шрифту виберіть на власний розсуд. Збільшіть панель групи.

8. Вставте у панель три прапорці (об'єкти типу CheckBox). Для цього використайте компоненту CheckBox ІХ із закладки Standard Палітри компонентів C++Builder. Вирівняйте прапорці, заздалегідь виокремивши їх (див. . 6) . Змініть значення властивості Caption (підпис) цих об'єктів на такі, як показано на рис . 22.2. Стиль і колір шрифтів виберіть на власний розсуд.

9. Встановіть прапорці "На екран" та "У масив" у положення "увімкнено". Для цього виокремте ці об'єкти та змініть значення їхніх властивостей Checked (контроль вибору) на True.

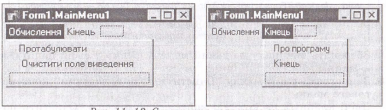
10. Вставте у форму багаторядкове поле редагувалня (об'єкт Memo). Для цього використайте компоненту Memo із закладки Standard палітри компонентів C++Builder. Збільшіть розміри поля. Властивість ScroiiBars (наявність смуг прокручування) цього об єкта задайте як ssBoth (будуть обидві смуги- вертикальна і горизонтальна) . File ~ Save all .

11 . Заоцайте початкові значення для полів редагування лівої і правої меж аргумента функЦії та для кроку зміни цього аргумента, наприклад, Так, як на рис. 9. Для цього змініть властивість Text цих обєктів. Для набору символа десяткової крапки використайте символ, передбачений операційною системою комп'ютера.

12. Витріть слово Memo1 у багаторядковому полі редагування (див. пояснення нижче). Для цього у вікні Object Inspector змініть значення властивості Lines (рядки) об'єкта Memo1 . Натиснувши на кнопку .:.:J, викличте вікно редагування цієї властивості (вікно String list editor - редактор багаторядкового поля). Витріть слово Memo1 та закінчіть роботу з цим вікном, клацнувши на кнопці Ok.

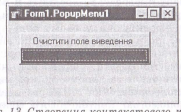
13. Вставте у форму головне і :контекстове меню (об'єкти типу MainMenu і PopupMenu). Для цього використаіі:те компоненти MainMenu 'ifl і PopupMenu із закладки Standard. Розташуйте піктограми в довільному місці форми, fta етапі виконання програми вони будуть не видимими.

14. Еведіть назви команд головного меню форми (див. рис.22.3 і рис. 22.4).Для цього виберіть об'єкт MainMenu1 і двічі клацніть назначенні його властивості ltems. Іиший шлях - двічі клациіть нa самому об'єкті . У вікні, яке відкриється (Form1->MainMenu1), вибирайте мишею рамку команди і записуйте назву команди, наприклад, "Обчислення", як ?Начення властивості Caption у вікні Object Ispector. Закрийте вікно створення команд головного меню FQrm1->MainMenu1.



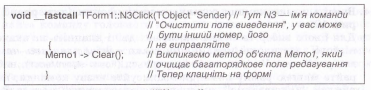
**Рис. 22.3,22.4. Створення головного меню**

15. Еведіть назви команд контекстового меню форми. Для цього аналогічно змініть значення властивості ltems об'єкта PopupMenu1 за допомогою вікна Forшl.PopupMenul (виклик цього вікна див. у п. 14). У вікні Object Ispector введіть текст "Очистити поле виведення" без лапок як значення властивості Caption (рис.22.5). За бажанням можете придумати ще якусь команду. Закрийте вікно Forш1.PopupMenu1 . Збережіть форму (Save АІІ).



**Рис. 22.5. Створення контекстового меню**

16. Запрограмуйте команду "Очистити поле виведення" головного меню, скориставшись методом Clear об'єкта Memo1 . Методи об'єкта - це набір функцій, які, аналогічно до властивостей, застосовують до об'єкта. Як і у випадку властивості, ім'я об'єкта від імені його метода відокремлюється крапкою. Виконайте команду меню "Очистити поле виведення", не запускаючи програму на виконання. З'явиться заготовка функції реакції на подію виклику цієї команди. У ній запишіть команду виклику методу Clear для очищення поля виведення об'єкта Memo1:



17. Запрограмуйте команду ":Кінець" головного меню, скориставись стандартною функцією exit().



Збережіть виконану на даний момент форму у своїй папці (File => Save all) .

18. Запрограмуйте команду "Очистити поле виведення'' контекс­

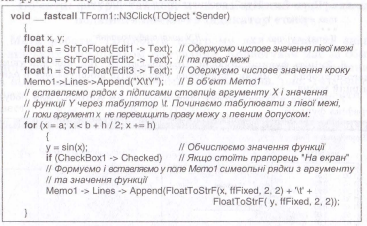
тового меню. Двічі клацніть на команді контекстового меню "Очистити поле виведення" у вікні Forш1.PopupMenu1 (виклик цього вікна див. у п. 14). Текст функції очищення такий самий, як і у команди головного меню, а саме: Memo1 -> Clear(); (див. п. 16).

19. "Прив'яжіть" контекстове меню PopupMenu1 до форми Form1. Клацніть fta формі і задайте властивість форми PopupMenu як PopupMenu1 .

Зауваження 2. Значення власт вості форми Menu автоматично вста новлюється як MainMenu1 у момент створен я голов ого меню (п . 13).

20. Запустіть створену програму та дослідіть ії роботу. Покспериментуйте з' багаторядковим полем редагування Memo1, вводячи та коректуючи у ньому будь-який текст. Зверніть увагу на те, що в цьому вікні можна виконувати такі ж дн з текстом як і в текстовому редакторі: виокремлювати фрагмент тексту, копіювати, переносити чи вилучати цей фрагмент. Витріть текст за допомогою команди головного меню "Очистити поле виведення". Ще раз введіть текст і витріть його за допомогою контекстового меню (для виклику контекs::тового меню форми потр бно клацнути правою клавішею миші на вільному місці форми). Закінчіть роботу програми, клацнувши на команді меню "Кінець".

21. Запрограмуйте команду "Протабулювати" . Властивість Lines об'єкта типу Memo є комплексною, тобто також є об'єктом зі своїми властивостями та методами. Результат роботи цієї програми - це таблиця, що складається з декількох рядків. Щоб долучити (додати) у поле Memo1 новий рядок до таблиці, треба змінити значення комплексної властивості Lines (рядки) за допомогою її методу Append (вставити) з одним аргументом - символьним рядком: Memo1->Lines->Append (рядок символів). Виконайте команду "Протабулювати" з головного меню, клацнувши на ній один раз. З'явиться заготовка функції, яку заповніть так:



22. Виконайте програму поекспериментуйте з різними значеннями лівої, правої межі та кроку аргументу. Закрийте вікно програми "Табулювання функції".

23. Збережіть створену програму у своїй папці.

24. Створіть ехе-файл вашої програми.

25. Закрийте С++ Builder, запустіть створену програму і виконайте обчислення для різних початкових даних.

26. Продемонструйте створену форму викладачеві. Закінчіть роботу.

**22.2. Створення навчальної програми**

**Вивчаємо англійську мову.**

**Об'єкти: фігура, таймер, мультимедійний програвач,подвійні кнопки та індикатор стану**

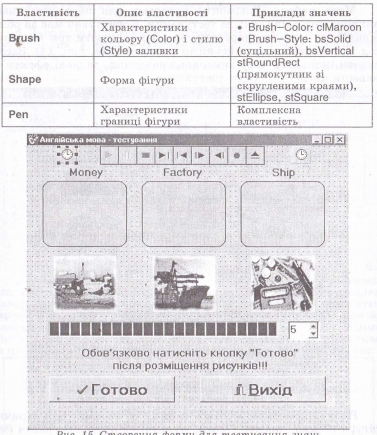
**Мета роботи**. Розробити програму для перевірки знань англійських слів шляхом тестування. Користувач має за обмежений час методом перетягування розташувати три малюнки під відпо'Відними англійськими словами (див. рис. 22.7). Застосувати індикатор часу виконання завдання, звукові ефекти та навести аналіз-підсумок тестування.



Набути павичків роботи з такими об'єктами: геометрична фігура (Shape), таймер (Timer), мультимедійний програнач (MediaPiayer), спарена кнопка з полем редагування (CSpinEdit), індикатори стану (Gauge та ProgressBar).

**Теоретичні відомості**.

Розглянемо деякі нові об'єкти. Таймер (Timer, піктограма С9 на закладці System) використовують для повторення фрагмента коду програми з певною періодичністю. Відповідний фрагмент розташовують у тілі функції опрацювання події OnТimer таймера. Періодичність вмикання таймера у мілісекундах задають властивістю lnteNal. Геометрична фігура (Shape, піктограма на закладці Additional) призначена для зображення елементарних геометричних фігур і має, зокрема, такі властивості:



**Рис. 22.8. Створення форми для тестувапня знань**

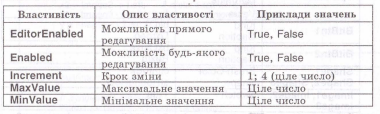
Мультимедійний програвач (MediaPiayer, піктограма на закладці System) призначений для програвання відео- та аудіофайлів. Керування програначем може здійснюватися як за допомогою традиційних кнопок РІау, Pause, Stop, Next тощо на етапі виконання програми, так і з програмного коду шляхом виконання методів цього об'єкта, наприклад:

MediaPiayer1->FileName="noвнe ім'я відео- чи аудіофайлу";

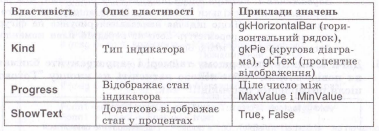
MediaPiayer1->0pen();

MediaPiayer1->Piay();

Спарена кнопка з полем редагування (CSpinEdit, піктограма на закладці Samples) призначена для введення та корекції цілочислового значення деякої величини під час роботи програми . Властивості кнопки:



Індикатор стану (CGauge, піктограма ІІІ" на закладці Samples, ProgressBar, піктограма 1'".r- на закладці Win32) використовують для наочної демонстрації стану виконання деякого процесу. Розглянемо три властивості індикатора CGauge:



**Хід роботи**

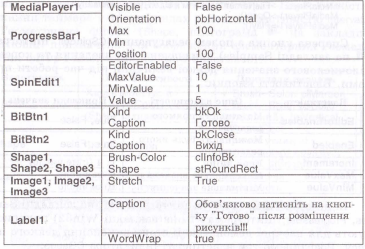
1. Завантажте середовище візуального програмування С++ Builder.

2. Замініть заголовок (Caption) форми з "Form1" на "Англійська мова - тестування".

3. Відмовтесь від усіх системних кнопок форми, задавши значення False в усіх позиціях властивості Borderlcons: biSystemMenu, biMinimize, biMaximize та ЬіНеІр. Роботу з програмою завершуватимемо натисканням на кнопку "Вихід".

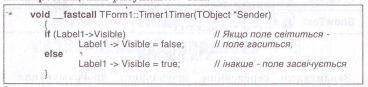
4 . Розташуйте на формі об'єкти так, як показано на рис. 22.8.





Зауваження 1. Розміри фігур повинні бути більшими, ніж розміри малюнків, оскільки малюнки слід розташовувати строго у середині фігур. Якщо під час накладання рисунка на фігуру рисунок зникає, перемістіть його на передній план командою його контекстового меню Bring То Front.

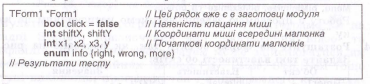
5. Двічі клацніть на першому таймері і запрограмуйте блимання повідомлення "Обов'язково натисніть на кнопку "Готово" після розміщення рисунків!!!" так:



Запустіть програму і переконайтесь, що напис на формі блимає.

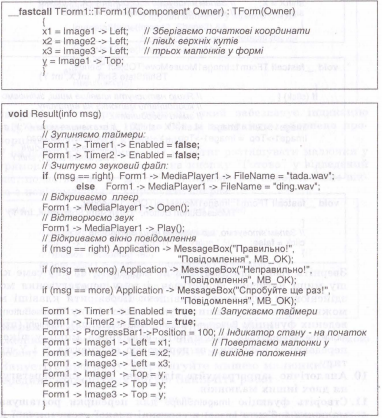
6. Уведіть опис глобальних змінних програми:

Для цього у вікні тексту програми після існуючого опису змінної форми введіть такі рядки:



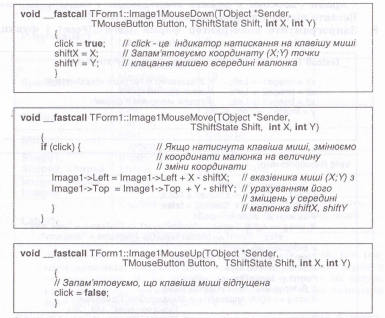
7. Вставте малюнки factory.bmp, shipping.bmp і finance.bmp з папки С:\ Program Files \Common Files \ Borland Shared \ Images Splash \ 256Color у об'єкти lmage1, lmage2 і ІmаgеЗ відповідно. Вставте підписи до цих малюнків (див. рис. 22.8).

8. Запрограмуйте конструн:тор форми TForm1::TForm1 і функцію Result() повідомлення результатів тесту:



Функція Result() не є методом класу Form1. Тому зміна властивостей будь-яких об'єктів форми в ній відбувається шляхом явного зазначення імені форми Form1 перед іменем відповідного об'єкта, наприклад, Form1->Timer1->Enabled.

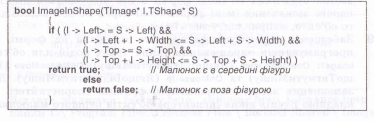
9. Забезпечте перетягування першого малюнка на формі, запрограмувавши опрацювання таких трьох подій для об'єкта lmage1: OnMouseDown (ЯкщоНатиснутиМишу), OnMouseMove (ЯкщоТягнутиМишу) та OnMouseUp (ЯкщоВідпуститиМишу). Для заповнення заготовок наведених функцій скористайтесь закладкою Events вікна інспектора об'єн:тів першого малюнка.



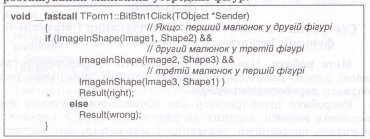
Зверніть увагу, що програма не перевіряє, на .яку саме клавішу миші натиснув користувач, і тому перетягування можна здійснювати будь-.якою клавішею. Перевірити клавіші миші Можна, проаналізувавши аргумент Button (типу TMouseButton) наведених функцій: Button = {mbleft ліва лавіша), mbRight (права ,mbMiddle (середня)} . Координати вказівника миші у кеелях передаються у функції за допомогою аргументів Х і У ці го типу.

10. Аналогічно запрограмуйте відповідні події для перетягування двох інших малюнків.

11. Створіть функцію lmagelnShape для перевірки розташування малюнка (об'єкта lmage) в середині деякої геометричної фігури (об'єкта Shape).

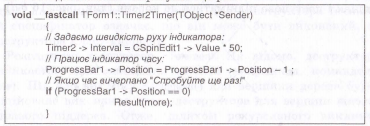


12. Запрограмуйте кнопку "Готово", яка перевіряє правильність розташування малюнків усередині фігур:



13. Запрограмуйте другий таймер, який забезпечує індикацію часу виконання від 100 до 0% зі швидкістю, обернено пропорційною до числа в CSpinEdit1.

У випадку, коли користувач не встиг розташувати малюнки у прямокутниках і натиснути на кнопку '"Готово" у відведений час, подається звук "ДІНГ" і відкривається інформаційне вікно з повідомленням "Спробуйте ще раз!".



14. Збережіть програму та проект у робочій папці.

15. Скопіюйте музичні файли ding.wav та tada.wav у свою робочу папку з папки С:\WINDOWS\MEDIA.

16. Запустіть програму. Перетягуйте мишею малюнки в середину відповідних фігур. Закінчіть роботу.