Лекція 4

Тема лекції: Програмування класів. Специфікатори доступу до членів класу. Використання функцій-членів. Створення та знищення об’єктів класу за допомогою конструкторів і деструкторів. Глобальні і локальні об’єкти класу. Масиви об'єктів класу. Почленна ініціалізація. Конструктор копії.

Класи

Клас - це вид структури, що об'єднує дані і функції. Наприклад:

```
class TTime
{
 public:
 int year;
 int month;
 int day;
 int hour;
 int minute;
 void Display(void);
};
```

Оголошення класу починається з ключового слова *class*. У дужках укладені 6 членів класу, яким передує слово *public*. Це специфікатор доступу. Він відчиняє доступ до всіх членів, що йдуть за ним, для всіх користувачів класу. Такі члени називаються відкритими.

Функції-члени класу оголошені прототипами функцій. Вони виконують деякі операції над даними-членами класу. Тіло функції розміщується в іншому місці програми. Імені функції-члена передує ім'я класу та оператор розширення області бачення. Наприклад:

```
void TTime::Display(void);
```

Тут заголовок функції вказує компілятору, що *Display()* - член класу *TTime*. У програмі можуть бути інші функції і функції-члени інших класів з ім'ям *Display()*, що не викликають конфліктів імен. Всередині функції-члена

оператори мають прямий доступ до членів класу.

Клас - це лише шаблон, схема, що описує формат членів класу. Щоб використовувати клас, необхідно створити об'єкт типу класу. Для доступу до членів об’єкту класу використовується оператор “. ”. Наприклад: використаємо клас *TTime*, що був описаний:

```
main()
{
 TTime appointment;       //Об'єкт типу TTime
 appointment.month = 7;
 appointment.day = 14;
 appointment.year = 2013;
 appointment.hour = 8;
 appointment.minute = 30;
 cout << “Appontment ==“;
 appointment.Display();  //Виклик функції-члена класу
 cout << ‘\n';
 return 0;
}
void TTime::Display(void)
{
 char s[32];
 sprintf(s,”Date:%02d/%02d/%04dTime:%02d:%02d\n”,month,day,yea
r,hour,
 minute);
 cout << s;
}
```

У програмі можуть бути інші об'єкти класу TTime:

```
TTime today;
TTime tomorrow;
TTime yesterday;
```

Цим об’єктам можна привласнити значення і потім відобразити їх за допомогою таких операторів:

```
today.Display();
tomorrow.Display();
yesterday.Display();
```

C++ дозволяє структурам і об'єднанням мати функції-члени подібно класам.

Структура - це клас, усі члени якого відкриті. На практиці функції-члени в структурах і об'єднаннях використовуються дуже рідко.

У розглянутому прикладі при зміні даних-членів класу необхідно буде змінювати оператори привласнення. Створимо новий клас, що полегшить зміну структури даних членів класу:

```
#include<iostream.h>
#include<stdio.h>
class TTime
{
 private:
 int year,int month,int day,int hour, int minute;
 public:
 void Display(void);
 void GetTime(int &m,int &d,int &y,int &hr, int &min);
 void SetTime(int m,int d,int y, int hr,int min);
};
main()
{
 TTime appointment;
 int month, day, year, hour, minute;
 appointment.SetTime(7,14,1996,8,30);
 cout << "Appointment==";
 appointment.Display();
 appointment.GetTime(month,day,year,hour,minute);
 appointment.SetTime(month,day,year,++hour,minute);
 cout << "Next hour==";
 appointment.Display();
 return 0;
}
void TTime::Display(void)
{  ..... ....}
void TTime::GetTime(int &m,int &d,int &y,int &hr, int &min)
{
  m = month;
```

```
  d = day;
  y = year;
  hr = hour;
  min = minute;
}
void TTime::SetTime(int m,int d,int y,int hr,int min)
{
 month = m;
 day = d;
 year = y;
 hour = hr;
 minute = min;
}
```

У даній програмі 3 доповнення:

Специфікатор доступу *private* робить дані-члени закритими. Тепер тільки функції-члени класу можуть безпосередньо посилатися на ці дані-члени. Тобто закриті члени невидимі поза класом. Члени класу закриті за замовчуванням. У класі можуть бути присутні відкриті і закриті члени, що розташовуються в довільному порядку, функції-члени можуть бути оголошені в закритій частині, але зазвичай вони відкриті.

Додані функції-члени *GetTime()* і *SetTime()*. З функції *GetTime()*значення компонентів об’єкту класу передаються назад, до аргументів оператора виклику, за допомогою параметрів-посилок. У *SetTime()* значення параметрів привласнюються даним-членам класу. Оскільки дані-члени класу *TTime* закриті, тому неможливо привласнювати значення безпосередньо об'єкту класу. Для використання закритих даних-членів класу існує єдиний засіб - виклик функцій-членів даного класу.

Більшість класів краще оголошувати в заголовочних файлах. Проте, це не є обов'язковим засобом. Приклад: оголошення класу *TTime* у заголовочному файлі *time1.h*:

```
class TTime
{
 private:
   int year;
```

```
    int month;
    int day;
    int hour;
    int minute;
  public:
    void Display(void);
    void GetTime(int &m, int &d, int &y, int &hr, int &min);
    void SetTime(int m, int d, int y, int hr, int min);
  };
```

Оголошення класу в заголовочному файлі дозволяє використовувати цей клас в інших модулях. Тіла функцій-членів наводяться в окремому файлі, що закінчується на .CPP. Включається створений заголовочнй файл у програму так:

```
#include "time1.h"
```

У більшості випадків виклики функцій-членів замість прямого доступу до даних-членів позначаються на швидкодії програми. Для досягнення найбільшої ефективності програм, C++ дозволяє оголошувати класи з функціями-членами, що вбудовуються. Хоча вбудовані функції-члени використовуються так само, як і інші функції, у скомпільованому коді вони не викликаються. Вони безпосередньо вбудовуються в скомпільовану програму. Наприклад:

```
class TTime
 {
  public:         ...... ....
  char *GetSTime(void)
  {
   char *cp=strdup(ctime(&dt));
   return cp;
  }
  void Display(void)
  {
   cout << ctime(&dt);
  }
 };
```

Тіла функцій-членів, що вбудовуються, вкладені у фігурні дужки відразу після заголовку функції в оголошенні класу. У даному випадку функції-члени,

що вбудовуються, оголошувалися безпосередньо в класі. Але можна їх оголосити й у модулі реалізації класу за допомогою ключового слова *inline*. Поставивши *inline* перед ім'ям функції, можна і не домогтися бажаного результату, оскільки реалізація функцій-членів, що вбудовуються, має бути доступна компілятору до того, як ця функція буде викликана.

Функції-члени, що вбудовуються найкраще розпізнаються, коли вони оголошені безпосередньо в класі. В оголошенні класу можна створити декілька функцій з однаковими іменами, що відрізняються хоча б одним параметром. Їхні тіла реалізуються окремо. Параметри функцій-членів за замовчуванням використовуються в оголошенні класів і не наводяться при виклику функції-члена.

Приклад оголошення функції-члена в класі:

```
void SetTime(int m=-1,int d=-1,int y=-1);    .... .
```

Реалізація функції-члена поза класом:

```
void TTime::SetTime(int m,int d,int y)
{
 //Тіло функції
}
```

Конструктори і деструктори

У класах може бути оголошено декілька конструкторів для автоматичної ініціалізації об’єкту класу при його створенні. Деструктор, призначений для очищення при виході об’єкту класу з області бачення, може бути тільки один. C++ автоматично викликає конструктори і деструктори. Конструктори оголошуються як функції-члени, які не повертають ніяких значень, із довільним числом параметрів будь-якого типу. Конструктори частіше оголошуються у відкритій секції, але можуть визначатися в будь-якому місці класу. Вони мають те ж саме ім'я, що і клас, для якого оголошуються. Наприклад:

```
class TTime
{
 private:
  long dt;
```

```
  char *dts;
  void DeleteDts(void);
 public:
  TTime();  //Конструктор
  TTime(int m,int d=-1,int y=-1); //Конструктор
  ~TTime(); //Деструктор
             ...
};
```

Конструктор, оголошений без параметрів, викликається за замовчуванням. Тут він оголошений першим. Другим оголошується перевантажений конструктор із трьома дійсними параметрами, два з яких передають значення за замовчуванням. Тіла конструкторів повинні знаходитися в модулі реалізації класу:

```
TTime::TTime()
{
 dts=NULL;    //Обнулення поточного рядка
 SetTime(-1,-1,-1);
}
TTime::TTime(int m,int d,int y)
{
 dts=NULL;
 SetTime(m,d,y);
}
TTime::~TTime()
{
 delete dts; //Видалення рядка, що зберігається в об'єкті
 dts=NULL;   //Обнулення покажчика
}
```

Програмування внутрішніх операторів конструкторів нічим не відрізняється від програмування інших функцій-членів. Але обов'язки конструктора обмежені привласненням початкових значень даним-членам класу, виділенням пам'яті, використаної об'єктом класу і т.і.

У даному випадку символьний покажчик *dts* встановлюється рівним *NULL*. Це означає, що покажчик ще не посилається на рядок. C++ автоматично викликає конструктор для ініціалізації об’єкту класу *TTime*, тому всі такі

об'єкти гарантовано мають ініціалізований покажчик *dts*. У будь-якому класі можна оголосити стільки конструкторів, скільки потрібно. Вони повинні відрізнятися хоча б одним параметром.

Приклад: використання конструкторів класу *TTime*:

```
main()
{
  TTime t1;
  TTime t2(8);
  TTime t3(8,1);
  TTime t4(8,1,2013);
  t1.Display();
  t2.Display();
  t3.Display();
  t4.Display();
  return 0;
}
```

Тут оголошуються чотири об'єкти класу *TTime*. C++ автоматично викликає конструктор класу при створенні об’єкту. Тут для *t1* викликається конструктор для ініціалізації об'єкту. Далі викликається конструктор із параметрами.

Деструктори зазвичай оголошуються у відкритій секції класу, але можуть і не бути оголошеними зовсім. Клас може мати не більше одного деструктора з ім'ям класу, у якому він оголошений. Перед оголошенням деструктора завжди ставиться знак ’~' (тільда). Якщо об'єкт класу виходить з області бачення, то C++ автоматично викликає деструктор об'єкту класу. Це дає об'єкту зручну можливість для самоочищення. Зазвичай явно деструктор не викликається.

Глобальні та локальні об'єкти класу. Покажчики на об'єкти

Кожна функція-член класу одержує прихований параметр з ім'ям *this*, що містить адресу об’єкту, для якого була викликана ця функція.

Об'єкти класу можуть бути глобальними або локальними у функції. Як і змінні звичайних типів, об'єкти класу можуть також адресуватися покажчиками

і посилками.

Природа об’єкту впливає на те, коли будуть викликатися конструктори і деструктори цього об’єкту класу. Конструктори глобальних об'єктів класу викликаються до початку виконання функції *main()*. Це забезпечує ініціалізацію усіх глобальних об'єктів до формального початку виконання програми. Наприклад, глобальне оголошення

```
TTime today;
```

створює глобальний об'єкт *today* класу *TTime*. До виклику *main()* C++ викликає для *today* конструктор *TTime* за замовчуванням. Оголошення:

```
TTime SomeTime(9,4,2013);
```

ініціалізує глобальний об'єкт *SomeTime* класу *TTime* конкретною датою. Наявність списку параметрів примушує C++ зайнятися пошуком у класі *TTime* конструктора, спроможного прийняти зазначені аргументи. Деструктори глобальних об'єктів класу викликаються в якості частини коду завершення програми. Автоматичні об'єкти класу, оголошені локальними у функції, створюються разом із викликом функції і знищуються після її завершення. Дані-члени автоматичного об’єкту класу зберігаються в купі. У функції

```
void AnyFunction(void)
{
 TTime now;
   ...
}
```

за замовчуванням викликається конструктор *TTime* () для ініціалізації об’єкта класу *now* щоразу, коли викликається функція. Коли функція завершить роботу, C++ викликає деструктор *TTime* для об’єкта *now*. Якщо викликається функція *exit()* для завершення програми, деструктори глобальних об'єктів класу викликаються як звичайно, а деструктори будь-яких існуючих автоматичних об'єктів, локальних у функції, не викликаються.

Покажчики можуть посилатися на динамічні об'єкти класу, простір для яких зазвичай виділяється в купі за допомогою *new*. Оголосимо покажчик *pToday* на об'єкт класу типу *TTime* таким чином:

```
TTime *pToday;
```

Це оголошення може бути як глобальним, так і локальним у функції. Як і всі покажчики, *pToday* необхідно ініціалізувати до його використання:

```
pToday = new TTime;
```

Тут за допомогою оператора *new* виділяється простір у купі для об’єкту типу *TTime*. Адреса першого байту цього об’єкту привласнюється *pToday*. C++ у цьому прикладі викликає конструктор об’єкта за замовчуванням. Для використання іншого конструктора:

```
pToday = new TTime(9,4,2013)
```

Покажчики на об'єкти класу використовуються так само, як і покажчики на об'єкти інших типів. Наприклад:

```
sp = (*pToday).GetSTime();
```

Відбувається привласнєння змінній *sp* результату функції-члена *GetSTime()* для об’єкта класу, на який посилається *pToday*. Оператор:

```
sp = pToday -> GetSTime();
```

виконує ті ж самі дії.

Деструктор динамічного об'єкту класу викликається при його знищенні. Оператор

```
delete pToday;
```

знищує об'єкт, на який посилається *pToday*, звільняючи пам'ять цього об'єкту. Безпосередньо перед руйнуванням об'єкту C++ викликає деструктор класу. Наприклад, у випадку з класами *TTime* деструктор може звільнити пам'ять, на яку посилається покажчик *dts*.

Об'єкти класу можуть оголошуватись як посилки.

Приклад:

```
TTime today;             //Глобальний об'єкт today
TTime &rToday = today;   //Посилка на today
```

Посилка *rToday* може використовуватися замість *today*. Наприклад:

```
rToday.Display();
today.Display();
```

Ці записи еквівалентні. Оскільки посилки посилаються на реально існуючі об'єкти, то при вході в область дії посилки конструктор класу не викликається. Відповідно не викликається і деструктор при виході з області дії посилки. Ці дії виконуються тільки при створенні і видаленні самого об’єкту.

Можна передавати об'єкти класу, покажчики на об'єкти класу і посилки на них у якості аргументів функцій. Приклад:

```
void AnyFunction(TTime t)
{
 t.Display();
}
```

Функція *AnyFunction()* має параметр *t* типу *TTime*. У функції викликається функція-член *Display()* для об'єкту *t*. Програма може оголосити об'єкт класу *TTime*:

```
TTime today;
```

і передати його за значенням функції *AnyFunction()*:

```
AnyFunction(today);
```

Функції також можуть використовувати в якості параметрів покажчики на клас. Передача функцією великих об'єктів класу за значенням призводить до виділення простору небажаного розміру. У таких випадках краще використовувати параметри-покажчики:

```
void AnyFunction(TTime *tp)
{
 tp ->Display();
}
```

Для виклику такої функції оператор виклику передає адресу об’єкту класу *TTime* замість самого об'єкту:

```
AnyFunction(&today);
```

Параметри функції можуть бути оголошені як посилки на об'єкти класу. Результати будуть аналогічні до попереднього випадку, тільки всередині функції не будуть потрібні операції доступу до покажчика:

```
void AnyFunction(TTime &tr)
{
 tr.Display();
}
```

У іншому місці програми можна передати об'єкт класу *TTime* безпосередньо в якості параметра-посилки AnyFunction():

```
AnyFunction(today);
```

Але такий оператор схожий на передачу аргументу *today* за значенням. Це

- недолік параметрів-посилок. Функції можуть повертати об'єкти класів безпосередньо, в якості покажчиків або як посилки. Найрідше використовується функція, що повертає безпосередньо об'єкт класу.

Приклад: функція, що виділяє у пам'яті новий об'єкт класу TTime та повертає його адресу:

```
TTime *newTime(void)
{
 TTime*p = new TTime;
 return p;
}
```

Всередині функції покажчику *p* на *TTime* привласнюється адреса об’єкта класу *TTime*, створеного за допомогою *new*. Покажчик існує тільки всередині функції, але об'єкт, на який посилається цей покажчик - глобальний. Можна використовувати функції, на які посилаються, для посилок на існуючі об'єкти:

```
TTime &newTime(void)
{
 return today;
}
```

У іншому місці можна оголосити посилку на *TTime* і привласнити їй результат *newTime()*:

```
TTime &tr = newTime();
```

Тепер посилка *tr* - псевдонім для *today* і може використовуватися в операторі вигляду:

```
newTime().Display();
```

Масиви об'єктів класу

Масиви об'єктів класу оголошуються так само, як і масиви даних звичайних типів. Наприклад:

```
TTime tenTimes[10];
```

Але існує правило: об'єкти класу, що будуть містити масив, повинні мати конструктори за замовчуванням. При створенні масивів об'єктів класу викликається конструктор класу за замовчуванням для кожного об’єкту масиву.

Не існує простого засобу ініціалізувати одні об'єкти класу з масиву за допомогою конструктора за замовчуванням, а інші - за допомогою альтернативного конструктора. У таких випадках оголошується масив, а потім ініціалізуються наново значення потрібних об'єктів заданим чином. Масиви об'єктів класу можуть також розміщуватись в купі й адресуватися за допомогою покажчиків. Приклад:

```
main()
{
  TTime *tarray;
  tarray = new TTime[6];
  for(int i=0;i<6;i++)
  tarray[i].Display();
  delete[] tarray;
  return 0;
}
```

Тут оголошується покажчик *tarray* на об'єкт класу *TTime.* Для ініціалізації покажчика викликається *new*, що виділяє в купі масив із 6 об'єктів і привласнюється адреса першого об’єкта масиву. У зв'язку зі зверненням до *new* 6 разів автоматично викликається конструктор за замовчуванням класу *TTime* для ініціалізації кожного об’єкта в масиві. Після завершення використання масиву об'єктів, його потрібно видалити. Для вказівки компілятору, що *tarray* посилається на масив в операторі *delete* використовуються порожні дужки. Така команда забезпечує виклик деструктора для кожного об’єкта в масиві.

Копіювання об'єктів класів. Конструктор копії

Коли один об'єкт класу копіюється в інший об'єкт сумісного типу, можуть утворитися найнесподіваніші результати. Особливо це стосується класів, в яких оголошені члени-покажчики на дані. Виникають такі небезпечні ситуації:

- Один об'єкт використовується при оголошенні нового об’єкта цього ж класу для його ініціалізації.
- Об'єкт передається функції параметром типу класу за значенням.
- Функція повертає об'єкт класу (але не посилку або покажчик на об'єкт).

- В операторі один об'єкт привласнюється іншому.

У перших трьох випадках відбувається створення та ініціалізація нової копії об’єкту існуючим об'єктом класу. У четвертому випадку один існуючий об'єкт привласнюється іншому, раніше оголошеному об'єкту. Існує різниця між копіюванням об’єкту і привласненням одного об’єкту іншому. У перших трьох випадках об'єкти створюються за допомогою виклику конструктора, у четвертому - привласнення одного об’єкту іншому не створює нового об’єкта і не призводить до виклику конструктора.

Копіювання об'єктів може призвести до появи покажчиків, що посилаються на те ж саме місце в пам'яті. Видалення одного з таких покажчиків призведе до того, що інші покажчики будуть посилатися на звільнену ділянку пам'яті. А видалення тієї ж самої ділянки пам'яті більше одного разу в деструкторі класу може призвести до руйнування купи. В C++ існує два механізми для безпечного копіювання і привласнення об'єктів класу, що мають члени-покажчики: почленна ініціалізація і почленне привласнення.

Коли один об'єкт використовується для ініціалізації іншого об’єкту класу, C++ копіює кожен член, що містить дані, з існуючого об'єкта до нового. Наприклад, розглянемо спочатку клас, що не має членів-покажчиків:

```
class TAnyClass
{
 private:
  int i;
  double r;
  char *s;
 public:
  TAnyClass()
  {
    i=0;
    r=0;
   }
  TAnyClass(int ii,double rr)
  {
   i=ii;
   r=rr;
```

```
  }
};
```

У класі оголошені два конструктори: конструктор за замовчуванням і параметризований конструктор, що ініціалізує об'єкт класу явними значеннями. Створимо об'єкти цього класу:

```
TAnyClass V1;
TAnyClass V2(100,3.14);
```

Ініціалізуємо новий об'єкт цього класу:

```
TAnyClass V3 = V2;
```

Це призведе до копіювання об’єкта *V2* у *V3* із почленною ініціалізацією даних-членів об’єкта *V3*. Тобто, дані-члени об’єкта класу *V2* копіюються один по одному у члени *V3*, що рівнозначно такому запису:

```
TAnyClass V3;
V3.i=V2.i;      //Для демонстрації
V3.r=V2.r;      //Оператори не скомпілюються, тому що i та r - закриті.
```

Почленна ініціалізація нового класу виконується і тоді, коли функції передається об'єкт за значенням і коли функція повертає об'єкт класу. Тепер введемо до класу закритий член-покажчик:

```
char *s;
```

Додамо також конструктор копії для створення об'єктів, що копіюються з інших об'єктів. Конструктор копії може виконувати будь-які дії, необхідні для створення безпечної копії, зокрема, не дублювати покажчики. У разі потреби C++ створює конструктори копії для класів, в яких вони не визначаються. Конструктор копії оголошується як *class(const class&)*. Як і всі конструктори, він не може бути віртуальним та успадкуватися похідним класом. Отже, нова версія *TAnyClass*:

```
class TAnyClass
{
 private:
  int i;
  double r;
  char *s;
 public:
```

```
  TAnyClass()
  {
    i=0;
    r=0;
    s=NULL;
   }
  TAnyClass(int ii, double rr, const char *ss)
   {
    i=ii;
    r=rr;
    s=strdup(ss);
    }
~TanyClass()
   {
    delete s;
   }
const char *GetStr(void)
   {
   return s;
   }
TAnyClass(TAnyClass &copy);
};
TAnyClass::TAnyClass(TAnyClass &copy)
{
 cout << "конструктор копії\n";
 i = copy.i;
 r = copy.r;
 if(copy.s)s=strdup(copy.s);
 else s=NULL;
}
```

Конструктор копії *TAnyClass(TAnyClass &copy)* має однакове ім'я з класом, у якому він визначений. У ньому оголошена посилка *&copy* на об'єкт класу, дані з якого копіюються до знов створеного об'єкту. Тут копіюються члени *i* та *r* з *copy* до нового об'єкту. Щоб можна було перетворити рядок, на який посилається покажчик s, за допомогою функції *strdup()*, виділяється ще одна ділянка пам'яті в купі для копії початкового рядка. Або ж, якщо в

покажчику s об’єкту *copy* утримується 0, покажчику нового об’єкту привласнюється *NULL*. Для скопійованих об'єктів класу *TAnyClass* немає небезпеки, що декілька покажчиків s будуть випадково посилатися на одну й ту ж саму ділянку пам'яті. В операторі:

```
TAnyClass V3=V2;
```

викликається конструктор копії.