Класи в мові С++

Характеризуючи тип даних, ми, перш за все, визначаємо спосіб збереження відповідних змінних та об'єм пам'яті, необхідний для них. Не менш важливим є набір допустимих операцій з даним типом. Для ***стандартних типів*** даних ця інформація закладена в компіляторі. Нові типи даних в мові С++ створюються через класи. Останні реалізують певну предметну абстракцію і зберігають не лише інформацію про об'єкт, а й визначають набір допустимих дій з ним. Інформація про об'єкт – це дані-члени класу, а дії реалізують функції-члени класу. І хоча синтаксис дозволяє різні можливості, зазвичай дані-члени класу визначають ***закритими***, а методи, які ними оперують, ***відкритими***.

Синтаксис визначення класу в С++.

```
class <тег_класу>
{
      // визначення даних-членів класу
private: // захищені члени класу – дані
   <тип_1> <ідентифікатор_1>;
   <тип_2> <ідентифікатор_2>;
...
      // декларації функцій-членів класу
public: //відкриті члени класу – функції
   <тип_результату> <ідентифікатор_1>
   (<параметри_функції>);
   <тип_результату> <ідентифікатор_2>
   (<параметри_функції>);
...
}; //   крапка з комою – обов'язкова!
```

Приклад.

```
// інформація про успішність студента
class Student
{
 private :
    char name [20]; // ім'я студента
    double av_mark; // середній бал
    double ex_mark; // бал за екзамен
 public :
    char * get_name () {return name;}
    double get_mark () {return av_mark;}
    double get_exam () {return ex_mark;}
};
```

Що нам наразі відомо про клас?

- Оголошення (декларація) класу схоже на оголошення структури та може містити дані-члени та методи-члени класу. Зауваження. Структури в C++ також можуть включати методи – це основна їх відмінність від структур С.
- У класі є власний (закритий) блок, доступ до елементів якого можливий лише для членів класу та загальний (відкритий) блок, елементи якого доступні для всіх частин програми. Як правило, в закритому блоці містяться дані класу, а відкритими є методи класу.
- Відкрита частина класу – це його загальнодоступний ***інтерфейс***, а закритий блок забезпечує ***інкапсуляцію*** – приховування інформації та гарантує її цілісність.
- Комплект даних-членів існує окремо для кожного екземпляру класу, методи – спільні для всіх екземплярів.
- Таким чином, у класі реалізовані базові принципи ООП – абстракція та інкапсуляція даних.

Яким чином можна створити та проініціалізувати екземпляр класу?

Оскільки доступ до даних-членів можливий лише для методів класу, очевидно, що ініціалізацію даних при створенні екземпляру треба покласти на одну з функцій-членів класу. Проте, треба бути впевненим, що така функція буде викликатись кожного разу, коли створюватиметься черговий екземпляр даного класу. Тому цей обов'язок покладається на компілятор, для цього йому потрібно розпізнавати необхідну функцію серед всіх методів-членів класу. Це означає, що така функція повинна мати зумовлене ім'я – ним є ім'я (тег) класу, а ця спеціальна функція називається ***конструктором*** класу. Ще однією особливістю конструктора (крім того, що його неможливо викликати безпосередньо – це прерогатива компілятора) є відсутність результату, не вказується навіть службове слово **`void`**.

Приклад (продовження).

```
// інформація про успішність студента
class Student
{
 …
 public :
 …
// Конструктор класу має аргумент за умовчанням
   Student (double av_mark_,   double ex_mark_,
   char * name_ = "Noname" )
   {
   av_mark = av_mark_;
   ex_mark = ex_mark_;
   strncpy (name, name_, sizeof(name));
   name [sizeof(name)-1] = '\0';
   cout << "Create Student " << name << endl;
   }
};
```

Тепер можливо створити екземпляр класу **Student** :

```
int main ()
{
// створюється та ініціалізується екземпляр
 Student st1 (30, 30);
 cout << "Student " << st1.get_name () <<
 " av_mark = " << st1.get_ get_mark () <<
 " exam = " << st1.get_exam () << endl;
// створюється та ініціалізується екземпляр
  Student st2 (60, 40, "Ivanov");
 cout << "Student " << st2.get_name () <<
 " av_mark = " << st2.get_ get_mark () <<
 " exam = " << st2.get_exam () << endl;
   system("PAUSE");
  return 0;
}
```

Крім того, можна створити динамічний екземпляр класу **`Student`** з допомогою операцій **`new`** та **`delete`**:

```
Student *ps = new Student (4, 4.5, "You");
ps -> show ();
delete ps;
```

Яким чином можна знищити екземпляр класу?

Якщо екземпляр класу був створений динамічно операцією **`new`**, то він має бути знищений операцією **`delete`**. В інших випадках екземпляр існує, доки не завершить роботу блок, де він був створений, і лише після цього екземпляр знищується. В будь-якому разі, для знищення екземпляру викликається спеціальний метод-член класу – деструктор. Якщо він не визначений в класі, то компілятор створить деструктор за умовчанням. Те саме стосується і випадку, коли у класі не створений конструктор. Деструктор має ім'я класу з префіксом ~ і не має ні типу результату, ні параметрів.

В мові C++ існує проблема витоку пам'яті, тому непотрібні динамічні екземпляри варто вчасно знищувати.

Приклад (продовження).

```
class Student
{
public :
// Конструктор класу
   Student (double av_mark_,   double ex_mark_,
   char * name_ = "Noname" )
   {
      …
   }
// Деструктор класу – реально в цьому класі
// в ньому потреби немає, це просто ілюстрація
   ~Student ()
   {
      cout << "Destruct Student " << name <<
      endl;
   }
};
```

Визначення методів-членів класу.

У попередніх прикладах методи-члени класу визначались безпосередньо в класі. Такі функції за умовчанням вважаються **`inline`**-функціями. Тому за винятком зовсім невеликих за обсягом коду функцій, їх визначають, як правило, поза класом. В такому випадку необхідно певним чином дати вказівку компілятору, що дана функція відноситься до певного класу. Для цього використовується оператор області видимості `::`. Якщо у попередньому прикладі конструктор лише задекларувати в класі, а визначити поза класом, то його повне ім'я буде наступним:

```
Student :: Student (double av_mark_,
  double ex_mark_, char * name_).
```

Приклад (продовження).

```
class Student
{
public :
 …
// Декларація конструктору класу
  Student (double av_mark_,   double ex_mark_,
   char * name_ = "Noname" );
 …
};
// Визначення конструктору класу
Student :: Student (double av_mark_,
   double ex_mark_, char * name_)
  {
  av_mark = av_mark_;
  ex_mark = ex_mark_;
  strncpy (name, name_, sizeof(name));
  name [sizeof(name)-1] = '\0';
  cout << "Create Student " << name << endl;
  }
```

Конструктор за умовчанням.

Так називають конструктор, який дозволяє створювати екземпляри класів з неявною ініціалізацією даних. Як вже зазначалось, такий конструктор автоматично створюється компілятором для класів, в яких не визначений власний конструктор. Проте, як тільки в класі визначається хоч один конструктор, такий автоматичний конструктор перестає діяти. В разі необхідності створення екземплярів без ініціалізації, варто визначити в класі конструктор за умовчанням. Найпростіший спосіб зробити це – перевантажити конструктор, або визначити умовчання для всіх його параметрів.

Приклад (продовження).

```
class Student
{
public :
// Декларація конструктору класу
   Student (double av_mark_,   double ex_mark_,
    char * name_ = "Noname" );
// Декларація конструктору за умовчанням
   Student ();
};
// Визначення конструктору класу
Student :: Student (double av_mark_,
    double ex_mark_, char * name_)
   {   …   }
// Визначення конструктору за умовчанням
Student :: Student ()
   {   …   }
```

Створення проекту.

Вдосконалимо попередній приклад, розмістивши фрагменти коду в різних файлах згідно прийнятому стилю ООП. Визначення класу помістимо у файл з розширенням **`.h`** (наприклад, **`student.h`**). При цьому використаємо команду препроцесора **`#ifndef`**, щоб позбутись повторного включення коду. Визначення функцій класу помістимо у файл з тим самим іменем, що й клас, і розширенням **`.cpp`** (**`student.cpp`**) і нарешті код, який використовує даний клас, тобто клієнтський файл, – у файл **`use_student.cpp`**.

Алгоритм створення проекту в середовищі DevCpp.

1. Створити окрему папку, в яку будуть поміщатись всі файли проекту.
2. Пункт меню Файл→Створити→Проект (створюється файл main.cpp – зберігти його під потрібним іменем, наприклад, **use_student.cpp** і помістити в нього код використання класу).
3. Файл→Створити→Вихідний файл (на питання “Додати в проект?” дати згоду). Зберігти файл під потрібним іменем (наприклад, **student.h**) і помістити в нього визначення класу. На початку файлу включити директиви **#ifndef STUD_H** та **#define STUD_H**. В кінці файлу включити директиву **#endif**
4. Файл→Створити→Вихідний файл (на питання “Додати в проект?” дати згоду). Зберігти файл під потрібним іменем (наприклад, **student.cpp**) і помістити в нього визначення функцій класу. На початку файлу не забути про директиву **#include "student.h"**. Така сама директива має бути і в файлі **use_student.cpp.**

Конструктор копіювання.

Якщо в продовження розглянутого прикладу створити функцію, що одержує екземпляр класу, а потім викликати її, можна одержати несподіваний результат.

```
void f (Student s)
{
  cout << s.get_name() << " " << s.get_mark()
       << " " << s.get_exam () << endl;
}
int main(int argc, char *argv[])
{
    Student *p = new Student ();
    f (*p);
    delete p;
  return 0;
}
```

В результаті виконання цього прикладу ми одержимо 1 повідомлення про створення екземпляру і цілих 2 – про знищення!

Конструктор копіювання.

Справа полягає в тому, що параметр типу **`Student`** передається у функцію **`f()`** за значенням, а отже, створюється, а потім знищується зі стеку. Тому і виникає зайвий виклик деструктора. Але конструктор при створенні локального екземпляру класу у стеку не викликався! Дійсно створенням екземплярів, які необхідно ініціалізувати значенням вже існуючого екземпляру займається інший конструктор – так званий конструктор копіювання. В даному прикладі був викликаний такий конструктор, створений компілятором. Він просто поелементно копіює даний екземпляр. Але це не завжди доречно, адже, можливо, необхідне виділення пам'яті для даних членів, тощо. Для явного визначення конструктора копіювання необхідно дотримуватись особливого синтаксису:

```
<ім'я_класу> (const <ім'я_класу> & );
```

Параметром конструктора копіювання є стала змінна-посилання на екземпляр класу. Його призначення – коректне створення копії екземпляру. Особливо важливо це у випадку, коли членами класу є вказівники, пам'ять під які виділяється оператором **`new`**. Адже тоді поелементне копіювання копіює вказівник (поверхневе копіювання), а не об'єкт, на який він посилається, – для цього необхідне глибоке копіювання.