Лекція 8

Об’єктно-орієнтоване програмування

Лектор: ***Розова Людмила Вікторівна***

План лекції 8

1. Простори імен
2. Директиви препроцесору

Матеріали курсу:
https://github.com/LRozova/oop1

Глобальний простір імен

В глобальному просторі імен **std** визначена вся бібліотека стандарту C++.

```
#include <iostream>
//Підключення глобального простору імен
using namespace std;
int value=4; // Глобальна змінна
int main()
{//Локальна змінна перекриває значення глобальної
    value = 8;
 // В цій точці коду value=8
    value++; //збільшується локальна змінна
    ::value--;//зменшується глобальна  завдяки оператору ::
    cout << "Global value: " << ::value << "\n";
    cout << "Local value: " << value << "\n";
    return 0;
}//Локальна змінна знищується, значення глобальної залишається value=3
```

Конфлікт імен

Кожна змінна, клас, функція повинна мати унікальне ім’я.

При спробі створити дві змінні з однаковим ім’ям, класи, функції з однаковими прототипами виникає конфлікт імен.

Конфлікт імен виникає при наявності двох однакових ідентифікаторів в однієї області видимості.

При збільшенні розмірів програм і кількості програмістів, які працюють над одним проектом, зростає вірогідність виникнення конфлікту імен .

Вихід: В С++ додана концепція **просторів імен**

Простори імен

Простори імен в C++ призначені для поділу глобально визначених ідентифікаторів на незалежні один від одного області.

В рамках одного простору імен ідентифікатори повинні мати унікальні імена. В рамках всього проекту імена ідентифікаторів можуть повторюватися.

Простір імен визначається за допомогою ключового слова **namespace** і **імені** простору, за якими в фігурних дужках слідує тіло простору імен.

Наприклад:

```
namespace FirstSpace   //заголовок простору
{    int a;
     int b;
     …
}
```

За межами простору імен доступ до ідентифікаторів, визначених у його тілі, може бути здійснений за допомогою **операції ::**, або при «підключенні» конкретних ідентифікаторів або всього простору за допомогою ключового слова **using** (діє в межах області видимості).

Наприклад:

```
FirstSpace::a=10;
```

або

```
using FirstSpace::a;    //using - оголошення
    a=10;
```

або

```
using namespace FirstSpace;    //using - директива
    a=10;
```

Розривне оголошення просторів імен

```
namespace   FirstSpace          //заголовок простору
    {
        int a;
        …
    }
namespace   FirstSpace          //заголовок простору
    {
        int b;
        …
    }
```

Все в одному просторі

Вкладеність просторів імен

Простори імен допускають вкладеність (***nesting***), тобто можуть містити інші простори імен.

Наприклад:

```
namespace Alpha
    {     int a;
   namespace Beta
       {
          double b;
       }
    }
```

```
int main()
 { using namespace Alpha;
   b=24.3;//Помилка: b – не визначено
   using namespace Alpha::Beta;
   b=24.3;//Вірно
   або    Alpha::Beta:: b=24.3;
```

C++ дозволяє створювати псевдоніми (синоніми) для просторів імен:

```
namespace Abeta=Alpha::Beta;
 Abeta::b=24.3;
```

Директиви препроцесору

Препроцесор - це частина компілятора, що піддає програму різним перетворенням до процесу компіляції. Програміст може давати препроцесору спеціальні команди (директиви), які здатні розширити дію середовища програмування.

Директиви - це спеціальні команди, які починаються з символу **#** і **НЕ** закінчуються крапкою з комою.

Препроцесор виконує аналіз тексту програми з урахуванням спеціальних директив.

Існує декілька типів директив.

Директиви препроцесору

 # (null directive)

Директива складається з одного символу #. ігнорується препроцесором

#include

Директива використовується для включення тексту заголовків файлів (* .h або * .hpp) бібліотек, що містять необхідні для роботи програми функції, типи, змінні і т.п., в текст програми або модуля.

#include <бібліотека>

Перший варіант використання (з кутовими дужками) застосовується для підключення стандартних бібліотек, шляхи для яких прописані в середовищі розробки.

Директиви препроцесору

#include “бібліотека”

Другий варіант застосовується для підключення бібліотек користувача.

Наприклад:

```
//підключення стандартної бібліотеки
#include <iostream>
//підключення математичної бібліотеки
#include <math.h>
//підключення заголовкового файлу
#include “sample.h”
```

Директиви препроцесору

#error

Директива видає помилку з текстом повідомлення, зазначеному в якості параметра. Використовується разом з умовною компіляцією.

```
#error повідомлення про помилку
```

#pragma

Залежить від конкретної реалізації компілятора

Макровизначення

Для макровизначень, або макросів використовується директива

#define

Ця директива визначає ідентифікатор (макроімен) і символьну послідовність, яка буде підставлена замість макроімені всюди, де воно зустрінеться в програмі.

#define макроім'я послідовність_символов

Наприклад:

```
#define UP 1
#define PR  “Привіт!”
#define MY_FAVORITE_NUMBER 9
```

Директива **#define і**з ідентифікатором, вже визначеним раніше іншою директивою #define, перекриває дію попередньої директиви.

В макроозначеннях можуть використовуватися службові символи:

**** - приєднує наступний рядок до визначення макросу

#undef

Використовується для видалення попереднього визначення деякого макроімені.

```
#undef  макроім'я
```

Наприклад:

```
#undef  PI
```

Умовна компіляція

Всі інші директиви препроцесору:

`#if, #ifdef, #ifndef,`

`#elif, #else, #endif`

використовуються для **умовної компіляції.**

Директиви препроцесора умовної компіляції дозволяють визначити, за яких умов код буде компілюватися, а при яких - ні.

Дозволяють проводити виборчу компіляцію частини вихідного коду.

Якщо вираз, що стоять після директиви **#if є істина**, то буде скомпільований код, що стоїть між директивою #if і директивою **#endif.**

Умовна компіляція

Наприклад:

```
#define MAX  10
int main()
{#if  MAX>10
   cout<<”Потребує додаткової пам’яті”;
 #else
     cout<<”Пам’яті достатньо”;
#endif
```

Умовна компіляція

#ifdef та **#ifndef**

Ці директиви пропонують ще два варіанти умовної компіляції, які можна виразити як «якщо визначено» і «якщо не визначено» відповідно.

```
#ifdef макроім'я1
//якщо визначено макроім'я через #define
    Последовательность инструкций 1
#endif

#ifndef макроім'я2 //якщо не визначено макроім'я
    Последовательность инструкций 2
#endif
```

Наприклад:

```
#ifndef MY_STRUCT  // якщо не визначено  MY_STRUCT
#define MY_STRUCT // визначемо і виконаємо далі програму
#endif  // кінець блоку перевірки
```

Або перевірка включення заголовкового файлу

```
#ifndef SOME_H_INCLUDED
#define SOME_H_INCLUDED
…
#endif
```

Наприклад:

```
x=3*z-7;
#ifdef Debug_Messages
     ShowMessage(String("x=")+x);
#endif
```

//якщо визначено Debug_Messages

```
#define Debug_Messages
```

// буде замінено препроцессором на такий код:

```
x=3*z-7;
ShowMessage(String("x=")+x);
```

// інакше в так:

```
x=3*z-7;
```

Дякую за увагу!