Практическое занятие 16

Тема: «Локальные и глобальные переменные»

Цель работы. Изучить принципы использования глобальных и локальных переменных.

Теоретическая часть

Переменные могут быть объявлены как внутри тела какой-либо функции, так и за пределами любой из них. Переменные, объявленные внутри тела функции, называются локальными. Такие переменные размещаются в стеке программы и действуют только внутри той функции, в которой объявлены. Как только управление возвращается вызывающей функции, память, отводимая под локальные переменные, освобождается.
Каждая переменная характеризуется областью действия, областью видимости и временем жизни. Под областью действия переменной понимают область программы, в которой переменная доступна для использования.
С этим понятием тесно связано понятие области видимости переменной. Если переменная выходит из области действия, она становится невидимой. С другой стороны, переменная может находиться в области действия, но быть невидимой. Переменная находится в области видимости, если к ней можно получить доступ (с помощью операции разрешения видимости, в том случае, если она непосредственно не видима). Временем жизни переменной называется интервал выполнения программы, в течение которого она существует. Локальные переменные имеют своей областью видимости функцию или блок, в которых они объявлены. В то же время область действия локальной переменной может исключать внутренний блок, если в нем объявлена переменная с тем же именем. Время жизни локальной переменной определяется временем выполнения блока или функции, в которой она объявлена.

Пример 1

```
#include  <iostream.h>
using  namespace std;
// Прототип функции
int  Sum(int a, int b) ;

int main()
{
// Локальные переменные
int x = 2;
int y = 4;
cout << Sum(x, y) ;
return 0; }

int Sum(int a, int b)
{
// Локальная переменная x
// видна только в теле функции Sum()
int x = a + b;
return x; }
```

Область действия глобальной переменной совпадает с областью видимости и простирается от точки ее описания до конца файла, в котором она объявлена. Время жизни

глобальной переменной - постоянное, то есть совпадает с временем выполнения программы.

Пример 2

```
#include <iostream>
using namespace std;

// Объявляем глобальную переменную Test
int Test = 200;

void PrintTest(void);

int main()
{
// Объявляем локальную переменную Test
int Test = 10;
// Вызов функции вывода глобальной переменной
PrintTest();
cout << "Локальная: " << Test << '\n';
return 0; }

void PrintTest(void)
{
cout << "Глобальная: " << Test << '\n';
}
```

В C++ допускается объявлять локальную переменную не только в начале функции, а вообще в любом месте программы. Если объявление происходит внутри какого-либо блока, переменная с таким же именем, объявленная вне тела блока, "прячется".

Пример 3

```
#include <iostream>

using namespace std;
// Объявляем глобальную переменную Test
int Test = 200;

void PrintTest(void);

int main()
{
// Объявляем локальную переменную Test
int Test = 10;
PrintTest();

cout << " Локальная 1: " << Test << "\n";
// Добавляем новый блок с еще одной
// локальной переменной Test
{
int Test=5;
```

```
cout << " Локальная 2: " << Test << "\n";}
// Возвращаемся к локальной Test вне блока

cout << " Локальная 1: " << Test << "\n";
return 0; }

void PrintTest(void)
{
cout << "Глобальная: " << Test << "\n";
}
```

C++ позволяет обращаться к глобальной переменной из любого места программы с помощью использования операции разрешения области видимости. Для этого перед именем переменной ставится префикс в виде двойного двоеточия (::).

Пример 4

```
#include  <iostream>
using  namespace  std;
// Объявление глобальной переменной
int  Turn = 5;

int  main(){
// Объявление локальной переменной
int  Turn = 70;
// Вывод локального значения
cout  <<  Turn  <<  endl;
// Вывод глобального значения:
cout  <<  ::Turn  <<  endl;
return  0;
}
```

Таблица 1. Модификаторы переменных

Модификатор	Применение	Область действия	Время жизни
auto	локальное	Блок	временное
register	локальное	Блок	временное
extern	глобальное	Блок	временное
static	локальное	Блок	постоянное
	глобальное	Файл	
volatile	глобальное	Файл	постоянное

Автоматические переменные

Модификатор auto используется при описании локальных переменных. Поскольку для локальных переменных данный модификатор используется по умолчанию, на практике его чаще всего опускают. Модификатор auto применяется только к локальным переменным,

которые видны только в блоке, в котором они объявлены. При выходе из блока такие переменные уничтожаются автоматически.

Пример 5

```
#include  <iostream>

using namespace std;

int main() {
auto int MyVar = 2;
// то же что int MyVar = 2;
cout << MyVar << endl;
return 0;}
```

Регистровые переменные

Модификатор register предписывает компилятору попытаться разместить указанную переменную в регистрах процессора. Если такая попытка оканчивается неудачно, переменная ведет себя как локальная переменная типа auto. Размещение переменных в регистрах, оптимизирует программный код по скорости, так как процессор оперирует с переменными, находящимися в регистрах, гораздо быстрее, чем с памятью. Число регистров процессора ограничено, поэтому количество таких переменных может быть очень небольшим. Модификатор register применяют только к локальным переменным. Попытка употребления данного модификатора (так же как и модификатора auto) к глобальным переменным вызовет сообщение об ошибке. Переменная существует только в пределах блока, содержащего ее объявление.

Пример 6

```
#include <iostream>
using namespace std;

int main() {
register int REG;
REG = 10;
cout << REG << endl;
return 0; }
```

Внешние переменные и функции

Если программа состоит из нескольких модулей, некоторые переменные могут использоваться для передачи значений из одного файла в другой. При этом некоторая переменная объявляется глобальной в одном модуле, а в других файлах, в которых она должна быть видима, производится ее объявление с использованием модификатора extern. Если объявление внешней переменной производится в блоке, она является локальной.
В отличие от предыдущих, этот модификатор сообщает, что первоначальное объявление переменной производится в каком-то другом файле.

Пример 7

```
// создайте файл myheader.h
// и внесите в него следующие строчки
//

bool Flag;
void ChangeFlag(void)
{
Flag = !Flag; // Значение переменной меняется на противоположное
}


// создайте файл myfunction.cpp
// там же, где находится  myheader.h
// и внесите в него следующие строчки

#include <iostream>
#include "myheader.h"

extern bool Flag;

using namespace std;

int main() {
	if(Flag)
		cout << "Flag is TRUE\n";
	else
		cout << "Flag is FALSE\n";

ChangeFlag();

	if(Flag)
		cout << "Flag is TRUE\n";
	else
		cout << "Flag is FALSE\n";

return 0; }
```

В файле myheader.h объявляется глобальная логическая переменная Flag и определяется реализация тела функции ChangeFlag(). В главном модуле подключается заголовочный файл myheader.h и переменная Flag описывается как внешняя (extern). Поскольку описание функции ChangeFlag() включается в главный модуль директивой #include "myheader.h", данная функция доступна в теле функции main().

Пространства имен

Определения функций и переменных в заголовочных файлах неразрывно связаны с понятием пространства имен. До введения понятия пространства все объявления идентификаторов и констант, сделанные в заголовочном файле, помещались компилятором

в глобальное пространство имен. Это могло приводить к возникновению конфликтов, связанных с использованием различными объектами одинаковых имен. Чаще всего это возникало при использовании в одной программе библиотек, разработанных различными производителями. Введение понятия пространства имен позволило значительно снизить количество подобных конфликтов. При включении в программу заголовочного файла его содержимое помещается не в глобальное пространство имен, а в пространство имен std.
Если в программе требуется определить некоторые идентификаторы, которые могут переопределить уже имеющиеся, следует создать новое пространство имен. Это достигается путем использования ключевого слова namespace. Объявления внутри нового пространства имен будут находиться только внутри видимости определенного имени пространства имен, предотвращая тем самым возникновение конфликтов. Например, можно создать следующее пространство имен:

```
namespace NewNameSpace
{
        int x, y, z;
        void SomeFunction(char smb);
}
```

В этом примере пространству имен NewNameSpace будут принадлежать переменные x, y, z, а также функция SomeFunction. Для того, чтобы указать компилятору, что следует использовать имена из конкретного именного пространства (в данном случае из NewNameSpace), следует использовать операцию разрешения видимости:

```
NewNameSpace::x = 5;
```

Однако, если в программе обращения к собственному пространству имен производятся довольно часто, то можно воспользоваться инструкцией using, синтаксис которой имеет две формы:

```
using namespace имя_пространства__имен;
или
using имя_пространства_имен::идентификатор;
```

При использовании первой формы компилятору сообщается, что в дальнейшем необходимо использовать идентификаторы из указанного именного пространства вплоть до того момента, пока не встретится следующая инструкция using. Например, указав в теле программы

```
using namespace NewNameSpace;
```

можно напрямую работать с соответствующими идентификаторами:

```
x=0;
y=z=4;
SomeFunction('A');
```

На практике часто после включения в программу заголовков явно указывается использование идентификаторов стандартного пространства имен:

```
using namespace std;
```

Следует понимать, что указание нового пространства имен инструкцией using namespace отменяет видимость стандартного пространства std, поэтому для получения доступа к соответствующим идентификаторам из std потребуется каждый раз использовать операцию разрешения видимости std::.

Пространства имен не могут быть объявлены внутри тела какой-либо функции, однако могут объявляться внутри других пространств. При этом для доступа к идентификатору внутреннего пространства необходимо указать имена всех вышестоящих именованных пространств. Например, объявлено следующее пространство имен:

```
namespace Highest
{
	namespace Middle
	{
		namespace Lowest:
		{
			int nAttr;
		}
	}
}
```

Использование объявленной переменной nAttr будет выглядеть:

```
Highest::Middle::Lowest::nAttr = 0;
```

Допускается объявление нескольких именованных пространств с одним и тем же именем, что позволяет разделить его на несколько файлов. Несмотря на это, содержимое всех частей будет храниться в одном и том же пространстве имен. Чтобы объявления переменных и функций в некотором пространстве имен были более упорядоченными, рекомендуется в пределах описания пространства имен объявлять только прототипы функций, помещая определение тела функции отдельно.

При этом следует явно указывать, к какому пространству имен относится функция:

```
namespace Nspace
{
	char c;
	int i;
	void Func1(char Flag) ;
}

void Nspace::Func1(char Flag)
{
	// тело функции
}
```

Допускается объявление не именованных пространств имен. В этом случае просто опускается имя пространства после ключевого слова namespace. Например:

```
namespace
{
	char cByte;
	long lValue;
```

```
}
```

Обращение к объявленным элементам производится по их имени, без какого-либо префикса. Не именованные пространства имен могут быть использованы только в том файле, в котором они объявлены.
Стандарт языка C++ предусматривает определение псевдонимов пространства имен, которые ссылаются на конкретное пространство имен. Чаще всего псевдонимы используются для упрощения работы с длинными именами пространств. Например, создание более короткого псевдонима и его использование для доступа к переменной:

```
namespace A_Very_Long_Name_Of_NameSpace
{
        float y;
}

        A_Very_Long_Name_Of_NameSpace:: y = 0.0 ;
        namespace
                Neo = A_Very_Long_Name_Of_NameSpace;
        Neo::y = 13.4;
```

Практические задания

1. Изучите примеры, приведенные в теоретической части.
2. Напишите программу решения квадратного уравнения, в которой используются как локальные, так и глобальные переменные. Дайте пояснения к ней.
3. Напишите программу которая использует оператор видимости.
4. Выполните задания практического занятия 15 с размещением функций во внешнем заголовочном файле. Напишите программу, которая демонстрирует их использование. Дайте пояснения к ней.
5. Выполните задания практического занятия 15 с использованием пространств имен и использование в нем переменных и функций. Дайте пояснения к ней.
6. Составьте отчет, в который включите результаты по пунктам 1-5. Приведите в отчет скриншоты результатов работы программ.