Занятие № 2.

Типы данных и основные структуры C++

1. Типы данных

Напомним, что основные типы а в C++:

int – целые числа

long – большие целые числа

double – дробные числа

string – строки

bool – логический тип: может принимать только значения true или false

Кроме того, есть еще типы данных:

char – один символ

signed char – число в диапазоне -127..127

unsigned char – число в диапазоне 0..255

wchar_t – расширенный символ для кодировок Unicode

unsigned int – неотрицательное целое число

float – дробные числа меньшей точности, чем double

void – тип без значения

Есть еще некоторые типы, которые редко используются на практике.

Заметим, что определить переменную типа void нельзя! Смысл этого типа мы покажем в будущем.

2. Массивы в C++

Основной структурой данных, без которой не обходится не одна программа – это массив. Массив представляет собой индексированный (числом) набор однотипных данных. Рассмотрим пример массива.

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
     int A[5];

     A[0] = 1;
     A[1] = 2;
     A[2] = 3;
     A[3] = 4;
     A[4] = 5;

     cout << A[2] << "\n";

     return 0;
}
```

В C++ массивы всегда начинаются с нуля. Поэтому при объявлении массива int A[5];

создается массив с элементами A[0], A[1], A[2], A[3], A[4] но нет A[5], потому что число в квадратных скобках указывает количество элементов.

При объявлении массива обычно желательно использовать константы. Константы представляют собой переменные, которые имеют тип данных и которым присваивается значение при объявлении, которое в дальнейшем уже нельзя изменить.

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
	const int N = 10; // объявить константу - количество элементов

	int A[N]; // создаем массив

	for (int n = 0; n < N; n++)
	{
		A[n] = n; // присваеваем массиву значения
	}

	int B = 0; // объявляем новую переменную

	for (int n = 0; n < N; n++)
	{
		B = B + A[n]; // суммируем
	}

	cout << "B = " << B << "\n"; // выводим результат

	return 0;
}
```

Задание: попробуйте изменить значение константы N. При этом использование константы позволяет менять это значение только в одном месте.

Для перебора массива можно использовать итераторы, например:

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
     const int N = 4; // объявить константу - количество элементов

     string Names[N]; // объявляем массив

     Names[0] = "Smith"; // заполняем массив
     Names[1] = "Ron";
     Names[2] = "Peter";
     Names[3] = "Ann";

     for (int i = 0; i < N; i++)
     {
          cout << Names[i] << "\n";
     }

     return 0;
}
```

3. Многомерные массивы

Многомерные массивы, это массивы, которые индексируются не одним числом, а двумя или большим числом индексов.

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
```

```
	const int N = 3; // объявить константу - количество элементов

	char T[N][N]; // создаем двумерный массив

	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < N; j++)
		{
			T[i][j] = 'O'; // заполняем массив ноликами
		}
	}

	for (int n = 0; n < N; n++)
	{
		T[n][n] = 'X'; // заполняем диагональ массива крестиками
	}

	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < N; j++)
		{
			cout << T[i][j] << "\t"; // выводим элементы массива
		}

		cout << "\n"; // после каждой строки новая строка
	}

	return 0;
}
```

Мы видим, что многомерный массив объявляется следующим образом

```
тип_данных имя_массива[количество_элементов]…[количество_элементов];
```

Задание: дополните предыдущую программу, чтобы в ней были отмечены крестиком обе диагонали.

4. Копирование массивов

Если у нас есть два массива одинаковой размерности A и B, то попытка присвоить

```
B = A;
```

приведет к ошибке. Поэтому для копирования массива A в массив B нужно применять операцию поэлементного копирования.

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
	const int N = 3; // объявить константу - количество элементов

	int A[N]; // создаем массив

	A[0] = 1;
	A[1] = 2;
	A[2] = 3;

	int B[N]; // создаем другой массив

	// B = A; // попытка присвоения массива - ошибка!

	for (int i = 0; i < N; i++)
```

```
	{
		B[i]  = A[i]; // копируем поэлементно
	}

	for (int i = 0; i < N; i++)
	{
		cout << B[i] << "\n"; // выводим значения копии массива
	}

	return 0;
}
```

Задание: напишите программу копирования двумерного массива.

5. Структуры

Массивы представляют собой последовательность переменных только одного типа. Для того, чтобы объединить переменные разного типа в C++ используются структуры.

Структура – это тип, группирующий переменные разного типа.

Рассмотрим пример использования структуры.

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
	struct TPerson // объявляем тип TPerson
	{
		string Name; // поля структуры
		int Year;
		double Height;
```

```
        bool Active;
    };

    TPerson Person; // объявляем переменную типа TPerson

    Person.Name = "John"; // заполняем поля структуры
    Person.Year = 1995;
    Person.Height = 175.4;
    Person.Active = true;

    cout << Person.Name << "\n"; // вывод значений структуры
    cout << Person.Year << "\n";
    cout << Person.Height << "\n";
    cout << Person.Active;

    return 0;
}
```

6. Массив структур

Структуры, как переменные, могут быть использованы для создания массивов.

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
{
    struct TItem // объявляем тип TPerson
    {
        string Name; // поля структуры
        int Number;
    };
```

```
	const int N = 3; // количество элементов в массиве

	TItem Items[N]; // объявляем массив структур

	for (int i = 0; i < N; i++) // запрашиваем структуры
	{
		cout << "Enter Name > ";
		cin >> Items[i].Name;
		cout << "Enter Number > ";
		cin >> Items[i].Number;
	}

	cout << "\n\n";
	for (int i = 0; i < N; i++)// печатаем структуры
	{
		cout << "Name : " << Items[i].Name << "\n";
		cout << "Number : " << Items[i].Number << "\n";
	}


	return 0;
}
```

7. Размер переменных

Для любой переменной можно узнать размер этой переменной в байтах. Для этого используется конструкция sizeof().

```
#include <iostream>

using namespace std; // используем пространство имен

int main()
```

```
{
	int a;
	double b;
	string s = "Mama";

	int A[10];

	struct TData
	{
		int x;
		double y;
	};

	cout << sizeof(a) << "\n";
	cout << sizeof(b) << "\n";
	cout << sizeof(s) << "\n";
	cout << sizeof(A) << "\n";
	cout << sizeof(TData) << "\n";
	cout << sizeof(bool) << "\n";

	return 0;
}
```

Заметим, что оператор sizeof() может применяться не только к переменным, но и к типам данных.