Севастопольский государственный университет
Кафедра информационных систем

Курс лекций по дисциплине

"АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ"

(АиП)

Лектор: Бондарев Владимир Николаевич

Лекция 6

Типы данных.
Концепция типов данных.
Структура программы.
Переменная. Выражения.

Концепция данных

Тип данных определяет:

✓ *множество допустимых значений*, которые может принимать переменная или константа обозначенного типа;

✓ *множество допустимых операций*, которые применяются к данным соответствующего типа;

✓ *способ хранения данных* в памяти компьютера.

Типы языка C++ делят на:

- основные;
- составные.

Основные – это типы данных для представления целых, вещественных, символьных и логических величин.

Составные – это типы, определенные пользователем, к ним относят массивы, перечисления, структуры, ссылки, указатели, объединения и классы.

Основные типы данных в C++

Основные (базовые, стандартные или арифметические) типы данных:

целочисленные:

- **`int`** (целый);
- **`char`** (символьный);
- **`wchar_t`** (расширенный символьный);
- **`bool`** (логический);

с плавающей точкой:

- **`float`** (вещественный);
- **`double`** (вещественный с двойной точностью).

Имеется также несколько **модификаторов**, которые можно использовать вместе с указанными базовыми типами.

- **`short`** (короткий);
- **`long`** (длинный);
- **`signed`** (знаковый);
- **`unsigned`** (беззнаковый).

Целый тип (int)

Размер типа не определяется стандартом, а зависит от архитектуры компьютера. Для 16-разрядного процессора отводится 2 байта, для 32-разрядного – 4 байта, для 64-разрядного – 8 байт.

short – требует под число 2 байта, **long** – 4 байта не зависимо от разрядности процессора. Поэтому возможно размер (**short int**) = размеру (**int**) и размер (**int**) = размеру (**long int**).

Описание переменных:

```
short int n;   или   short n;
long int k;    или   long k;
```

размер short<= размер int<= размер long

По умолчанию все целые знаковые, поэтому **signed** можно опускать.

35 – тип? размер?

Можно явно указать тип константы суффиксами **L,l,U,u**.

32L – занимает 4 байта

32Lu – ?

Целый тип (int)

Особенность:

внутреннее представление величины целого типа – число в двоичном коде и для спецификатора **signed** старший бит интерпретируется как знаковый (0 – положительное число, 1 – отрицательное), а для **unsigned** – только положительные числа, поскольку старший разряд рассматривается как код числа.

Диапазон значений величин целого типа зависит от спецификатора.

```
short int n; или short n;
long int x; или long x;
signed int y; или signed y;
unsigned int t; или unsigned t;
```

Символьный тип (char)

Значениями этого типа являются элементы упорядоченного множества символов. Значения такого типа обозначаются одним символом, заключенным в апострофы: **`'A','0','9','*',' '.`**

Пример описания в программе: **`char x;`**

Под величину символьного типа отводиться количество байт, достаточное для размещения любого символа из набора символов данного компьютера. Как правило, это 1 байт.

`signed char -128 … 127`

`unsigned char 0 … 255`

Примеры ASCII кодов некоторых литер:

```
'A'     65
'B'     66
'a'     97
'b'     98
'0'     48
'1'     49
```

Расширенный символьный тип (wchar_t)

Предназначен для работы с набором символов, для кодировки которых недостаточно 1 байта, например **Unicode**.

Размер этого типа соответствует типу **short** (как правило).

Строковые константы типа **wchar_t** записываются с префиксом L

```
L"something"
```

Логический тип (bool)

Величины такого типа могут принимать только значения **true** и **false**. Внутренне представление **false** – 0 (ноль). Любое другое значение интерпретируется на **true**. При преобразовании к целому типу имеет значение 1.

В Си ***отсутствует*** логический тип данных. Вместо него используется целый тип **int**:

0 – это ложь;

все, отличное от нуля – это истина.

Чаще всего истина – это 1, например вечный цикл **while** выглядит так:

```
while(1) {
        //тело цикла
};
```

Типы с плавающей точкой (float, double, long double)

Стандарт определяет три типа данных для хранения вещественных значений: **`float, double, long double`**.

Внутреннее представление состоит из двух частей: мантиссы и порядка. Пример описания в программе: **`float x; double y;`**

Тип	Диапазон значений	Размер (байт)
`bool`	true и false	1
`signed char`	-128 …127	1
`unsigned char`	0 … 255	1
`signed short int`	-32 768 … 32 767	2
`unsigned short int`	0 … 65 535	2
`signed long int`	-2 147 483 648 …2 147 483 647	4
`unsigned long int`	0 … 4 294 967 295	4
`float`	3.4e-38 … 3.4e+38	4
`double`	-1.7e+308 … 1.7e+308	8
`long double`	3.4e-4932 … 3.4e+4932	10

Соотношение размеров основных типов данных

Для написания переносимых на различные платформы программ ***НЕЛЬЗЯ*** делать предположений о размере типа **int**. Для его получения пользуются операцией **sizeof(int)**, результат которой типа байт.

Например, для **ОС MS-DOS sizeof(int)** вернет 2 байта, для **Windows 9X** – 4 байта.

В стандарте **ANSI** диапазоны значений основный типов не задаются, определяется только соотношение между размерами:

```
sizeof(float)<= sizeof(double)<=sizeof(long double)

sizeof(char)<= sizeof(short)<=sizeof(int)<=sizeof(long)
```

Максимальные и минимальные допустимые значения для целых типов зависят от реализации и приведены в заголовочном файле **<limit.h> (<climits>)**, для вещественных – в файле **<float.h> (<cfloat>)**

Тип void

Множество значений этого типа пусто.

Тип используется:

- для определения функций, которые не возвращают значений;
- как базовый тип указателей;
- в операции преобразования типов.

Подробнее эту тему рассмотрим дальше.

Структура программы

Программа на языке C++ состоит из функций, описаний и директив препроцессора. Одна из функций должна иметь имя **main**.

Выполнение программы начинается с первого оператора этой функции.

Простейшее определение функции имеет формат:

```
<Тип> <имя> ([<параметры>]) {
 <тело функции>
}
```

Тип – тип возвращаемого значения;

имя – название функции;

параметры – аргументы функции;

тело – операторы функции.

- Если функция ничего не возвращает, то тип **void**;
- тело функции является блоком и заключается в **{}**;
- функции ***не могут*** быть вложенными;
- каждый оператор должен заканчиваться **;** (кроме составного оператора).

Структура программы

Структура программы:

1 способ:

```
<директивы процессора>
<описания>
int f1(){
 <операторы функции f1>
}
int f2(){
<операторы функции f2>
}

int main(){
 <операторы главной функции>
}
```

2 способ:

```
<директивы процессора>
<описания>
int f1();
int f2();
int main(){
 <операторы main>
}
 int f1(){
 <операторы функции f1>
}
int f2(){
<операторы функции f2>
}
```

Программа может состоять из нескольких модулей (исходных файлов). Работу программы, состоящей из нескольких модулей рассмотрим позже.

Структура программы

Пример программы на С:

```
#include <stdio.h>
int main(){
printf("\nhello world\n");
return 0;
//  код никогда не будет выполнен
}
```

Директива препроцессора **#include <stdio.h>** обеспечивает включение в текст программы файла **<stdio.h>** – стандартной библиотеки ввода-вывода.

Пример программы на С++:

```
#include <iostream>
using namespace std;
int main() {
    cout<<endl<<"hello world"<<endl;
    return 0;
}
```

Переменная. Выражения.

В любой программе требуется производить вычисления. Для вычисления значений используются ***выражения***.

Выражения состоят из

- операндов,
- знаков операций;
- скобок.

Каждый операнд является, в свою очередь, выражением или одним из его частных случаев, например, константой или переменной.

Любое выражение, завершающееся точкой с запятой, рассматривается как оператор, выполнение которого заключается в вычислении выражения. Частным случаем выражения является пустой оператор `;` (он используется, когда по синтаксису оператор требуется, а по смыслу – нет).

Примеры:

```
i++;  // выполняется операция инкремента
a *= b + c;  // выполняется умножение с присваиванием
fun(i, к);  // выполняется вызов функции
```

Переменная. Выражения.

Переменная – это именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя и значение.

Имя служит для обращения к области памяти, в которой хранится значение. Во время выполнения программы значение переменной можно изменять. Перед использованием любая переменная должна быть описана.

Пример:

```
int a; float x;
```

Общий вид оператора описания переменных:

```
[класс памяти] [const] тип имя [инициализатор];
```

Необязательный класс памяти может принимать одно из значений: **`auto, extern, static и register.`**

Модификатор **`const`** показывает, что значение переменной изменять нельзя. Такую переменную называют именованной константой, или просто константой.

```
const int N=10, M=30;
```

Переменная. Выражения.

При описании можно присвоить переменной начальное значение, это называется инициализацией. Инициализатор можно записывать в двух формах – со знаком равенства (C) или в круглых скобках (C++):

```
int x=10, y(10);
```

В одном операторе можно описать несколько переменных одного типа, разделяя их запятыми.

Примеры:

```
short int a = 1;  // целая переменная a
const char C = 'C' ;  // символьная константа C
char s, sf = 'f' ;  // инициализация относится только к sf
char t(54); float c=0.22, x(3), sum;
```

Если тип инициализирующего значения не совпадает с типом переменной, выполняются преобразования типа по определенным правилам.

Переменная. Выражения.

Описание переменной, кроме типа и класса памяти, явно или по умолчанию задает ее область действия. Класс памяти и область действия зависят не только от собственно описания, но и от места его размещения в тексте программы.

Область действия идентификатора – это часть программы, в которой его можно использовать для доступа к связанной с ним области памяти. В зависимости от области действия переменная может быть локальной или глобальной.

Если переменная определена внутри блока она называется ***локальной***, область ее действия – от точки описания до конца блока, включая все вложенные блоки.

Если переменная определена вне любого блока, она называется ***глобальной*** и областью ее действия считается файл, в котором она определена, от точки описания до его конца.

Переменная. Выражения.

Класс памяти определяет ***время жизни*** и ***область видимости*** программного объекта (в частности, переменной). Если класс памяти не указан явным образом, он определяется компилятором исходя из контекста объявления.

Время жизни может быть постоянным (в течение выполнения программы) и временным (в течение выполнения блока).

Областью видимости идентификатора называется часть текста программы, из которой допустим обычный доступ к связанной с идентификатором областью памяти.

Чаще всего область видимости совпадает с областью действия. Исключением является ситуация, когда во вложенном блоке описана переменная с таким же именем. В этом случае внешняя переменная во вложенном блоке невидима, хотя он и входит в ее область действия. Тем не менее к этой переменной, если она глобальная, можно обратиться, используя операцию доступа к области видимости `::`.

Классы памяти

`auto` – автоматическая переменная. Память под нее выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти происходит при выходе из блока, в котором описана переменная. Время ее жизни – с момента описания до конца блока. Для глобальных переменных этот спецификатор не используется, а для локальных он принимается по умолчанию, поэтому задавать его явным образом большого смысла не имеет.

`extern` – означает, что переменная определяется в другом месте программы (в другом файле или дальше по тексту).

`static` – статическая переменная. Время жизни – постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменные могут быть глобальными и локальными. Глобальные статические переменные видны только в том модуле, в котором они описаны.

Классы памяти

register – аналогично **auto**, но память выделяется по возможности в регистрах процессора. Если такой возможности у компилятора нет, переменные обрабатываются как **auto.**

Если при определении начальное значение переменных явным образом не задается, компилятор присваивает глобальным и статическим переменным нулевое значение соответствующего типа. Автоматические переменные не инициализируются.

Описание переменной может выполняться в форме **объявления или определения. Объявление** информирует компилятор о типе переменной и классе памяти, а **определение** содержит, кроме этого, указание компилятору выделить память в соответствии с типом переменной. Большинство объявлений являются одновременно и определениями.

Классы памяти

```
  int a;              //1 глобальная переменная a

int main(){
  int b;              // 2 локальная переменная b
  extern int x;       // 3 переменная x определена в другом месте
  static int c;       // 4 локальная статическая переменная c
  a = 1;              // 5 присваивание глобальной переменной
  int a;              // 6 локальная переменная a
  a = 2;              // 7 присваивание локальной переменной
  ::a = 3;            // 8 присваивание глобальной переменной (C++)
}
```

Классы памяти

В этом примере глобальная переменная **a** определена вне всех блоков. Память под нее выделяется в сегменте данных в начале работы программы, областью действия является вся программа. Область видимости – вся программа, кроме строк 6-8, так как в первой из них определяется локальная переменная с тем же именем, область действия которой начинается с точки ее описания и заканчивается при выходе из блока. Переменные **b** и **c** – локальные, область их видимости - блок, но время жизни различно: память под **b** выделяется в стеке при входе в блок и освобождается при выходе из него, а переменная **c** располагается в сегменте данных и существует все время, пока работает программа.