

Данные в языке C

Виды данных

Переменная - это каким-либо образом поименованная и/или адресованная область физической или виртуальной памяти, предназначенная для хранения данных (значений).

Перечисление - это набор именованных целочисленных констант, определяющий все допустимые значения, которые может принимать переменная.

Константа - способ адресации данных, изменение которых рассматриваемой программой не предполагается или запрещается.



Вид \ Подвид	Числовая	Логическая	Символьная	Адресная
Константа	3, -23.76		'A', "Hello"	
Переменная	int p = 4;	bool b = true;	char c = 'd';	int a = 3; int *p = &a;
Перечисление				

Тип переменной

Тип требуется компилятору для того чтобы он мог:

- Выделить нужное количество бит для хранения значения переменной;
- Распознать в каком виде использует данные программист и перевести их в двоичное представление;(знак./беззнак.)
- При выполнении действий с переменными разного типа сгенерировать разные низкоуровневые команды:

```
int x1=1, y1 = 2, z1;  
z1 = x1 + y1;  // ->add
```

```
double x2=1.0, y2 = 2.0, z2;  
z2 = x2 + y2;  // ->fadd
```

Типы данных

Типы данных	Размер в байтах	Диапазон значений
unsigned char	1	от 0 до 255
signed char	1	от -128 до 127
unsigned int	2 или 4	от 0 до 65535 от 0 до 4 294 967 295
signed int	2 или 4	от -32768 до 32767 от -2 147 483 648 до 2 147 483 647
unsigned short	2	от 0 до 65535
signed short	2	от -32768 до 32767

Типы данных

Типы данных	Размер в байтах	Диапазон значений
long int	4	от -2 147 483 647 до 2 147 483 647
long long int	8	от $-(2^{63}-1)$ до $(2^{63}-1)$
unsigned long int	4	от 0 до 4 294 967 295
unsigned long long int	8	от 0 до $(2^{64}-1)$
float	4	от $\pm 3.4E-38$ до $3.4E+38$
double	8	От $\pm 1.7E-308$ до $1.7E+308$
long double	10	От $\pm 3.4E-4932$ до $1.1E+4932$
void		тип без значения

Приведение типов

Приведение типов может быть:

Неявное (компилятор)

Явное (программист)

Процессор не умеет работать с операндами разного типа. Прежде чем производить вычисления, компилятор должен все значения привести к одному типу. **!**

В ряде случаев преобразования сопровождаются потерей информации. Без потери информации проходят следующие цепочки преобразований:

char -> short -> int -> long

unsigned char -> unsigned short -> unsigned int -> unsigned long

float -> double -> long double

Приведение типов

При арифметических операциях компилятор преобразует:

1. Если имеется в выражении тип **long double**, то всё к типу **long double**
2. Если п.1 не выполняется и если имеется тип **double**, то всё к типу **double**
3. Если п.2 не выполняется и если имеется тип **float**, то всё к типу **float**
4. Если п.3 не выполняется и если имеется тип **unsigned long int**, то всё к типу **unsigned long int**
5. Если п.4 не выполняется и если имеется тип **long**, то всё к типу **long**
6. Если п.5 не выполняется и если имеется тип **unsigned**, то всё к типу **unsigned**
7. Если п.6 не выполняется то оба операнда приводятся к типу **int**

Правило неявного приведения:

- Определяется тип выражения справа от «=»;
- Значение выражения справа приводится к типу слева от «=».

```
int x = 1;  
double y = 2.2  
int z = x + y;
```

Приведение типов

Явное приведение типа (программист).

- Стил С:

```
int x = 1, y = 2;  
double z = (double) x/y;
```

- Стил С++:

static_cast <type>(выр-е) – преобр-е с проверкой корректности во время компиляции.

reinterpret_cast <type>(выр-е) – преобр-е без проверки во время компиляции.

const_cast <type>(выр-е) – константное преобр-е во время компиляции (аннулирует или назначает действие модиф.: **const, volatile**).

dynamic_cast <type>(выр-е) – преобр-е с проверкой во время выполнения

Достаточно явно привести к “старшему” типу только один операнд. Второй операнд будет приведен к “старшему” типу самим компилятором.

```
int d = static_cast < int >( 7.5 );  
  
int const * cpd = &d ;  
  
int * pd = const_cast < int * >( cpd );  
  
float f = static_cast < float > ( d );  
  
float * pf = reinterpret_cast<float*>( pd );  
  
float * pf2 = static_cast<float*>( static_cast<void*>(pd) );  
  
float h = *reinterpret_cast<float*>( &d );
```


Размещение и Время существования.



- Имеется 4 основных способа размещения переменных.
- От способа размещения зависит время жизни переменной.

**static auto register
malloc() new**

Объявление (declaration)

- Инструкция компилятору как использовать указанное имя. Описывает свойства переменной с указанным именем.
- **Переменная** – именованная область памяти, используемая для хранения информации. Свойства переменной определяются:
 - Заданием типа переменной (int, double, char)
 - Контекстом определения (где определена, с какими доп. ключ. словами)

```
static int number;  
extern int ptr;  
char ch;
```
- Встретив объявление переменной, компилятор :
 - Запоминает соответствие имени перемен. и типа (для контроля типов, для приведения типов);
 - Выделяет требуемый объем памяти для ее хранения (если еще есть и определение!) и ассоциирует в дальнейшем имя и адрес выделенного участка памяти;

Определение (definition)

- Определение «чего-то» означает предоставление всей необходимой информации для создания «этого» целиком. Определение функции означает предоставление тела функции, определение класса означает предоставление всех методов и полей класса.
- Существует правило единственного определения. Определение «чего-то» может быть только одно!

```
int func();    // объявление
int main()
{
    int x = func();
}
int func()    // определение
{
    return 2;
}
```

Инициализация

Копирующая
инициализация
`int n = 5;`

Прямая
инициализация
`int n(5);`

Uniform
инициализация
`int n {5};`

Инициализация –
присваивание значения при
определении.

`static = 0;`
`auto = NoN;`
`dynamic = NoN`

Область видимости. Соккрытие переменной.

- **Область видимости** - это те части программы, в которой пользователь может изменять или использовать переменные в своих нуждах.
- Если переменная была создана в блоке {}, то ее областью видимости будет являться этот блок от его начала до его конца включая все дочерние блоки созданные в нём.

```
....  
{  
    int i;  
    i++;  
}  
  
i = 1; // переменная не определена
```

```
int x = 5; // глобальные переменные  
int main()  
{  
    x = 1; // обращение к глоб.перем.  
    int x; // создание внешн. лок. перем.  
    x = 2; // внешн. лок. перем. присвоили 2  
    {  
        int x; // создание внутр. лок. перем.  
        x = 3;; // внутр. лок. перем. присвоили 3  
    }  
    x = 4;  
    return 0;  
}  
  
// мы не сможем обращаться к глоб. перем во  
// время работы ф-ции main(). Произошло  
// СОКРЫТИЕ ПЕРЕМ. x
```

Разрешение области видимости

- В C++, но не в C к скрытому глобальному имени можно обратиться с помощью оператора разрешения области видимости «::» !

Не существует способа обращения к скрытой локальной переменной !

```
int x;  
int main()  
{  
    x = 1;  
    int x;  
    ::x = 100;  
    x = 2;  
    {  
        int x;  
        ::x = 200;  
        x = 3;  
    }  
    x = 4;  
}
```

На сегодня всё !