



Ст. преп.  
каф. ПОВТ  
Масленников  
Алексей  
Александров  
ич



## Лекция № 3 Введение в программирование





Приведение типа позволяет перевести переменную из одного типа в другой.

Синтаксис приведения типов в языке C (и в большинстве современных языков):

(тип) выражение



Выделяют два основных типа приведения



**Явное – задается программистом вручную в коде (как на предыдущем слайде)**



**Неявное – выполняется транслятором (компилятором или интерпретатором) по правилам, описанным в стандарте языка**



Примеры неявного приведения типов:

```
double d; // вещественный тип
long   l; // целый тип
int    i; // целый тип

if ( d > i )    d = i;
if ( i > l )    l = i;
if ( d == l )  d *= 2;
```



При выполнении операций сравнения и при присваивании  
При неявных преобразованиях возможны побочные  
эффекты.



Например, при приведении числа вещественного типа к  
целому типу дробная часть отсекается (округление не  
выполняется).

При обратном преобразовании возможно понижение  
точности из-за различий в представлении вещественных и  
целочисленных чисел.



Например, в переменной типа `float` (число с плавающей точкой одинарной точности по стандарту IEEE 754), нельзя сохранить число 16 777 217 без потери точности, а в 32-х битной переменной целого типа `int` — можно.



Из-за потери точности операции сравнения одного и того же числа, представленного целым и вещественным типами (например, `int` и `float`), могут давать ложные результаты (числа могут быть не равны).



```
#include <stdio.h>

int main(void) {

    int i_value = 16777217;
    float f_value = 16777216.0;
    printf( "Значение integer :%d\n", i_value );
    printf( "Значение float : %f\n", f_value );
    printf( "Значение равны:%s\n", i_value == f_value?"Да":"Нет" );
    return 0;
}
```

Результат:

```
Значение integer :16777217
Значение float : 16777216.000000
Значение равны: Да
```



Пример явного приведения типов:

```
int X;  
int Y = 200;  
char C = 30;  
X = (int)C * 10 + Y;  
printf("X=%d",X);
```

Результат:

```
X=500
```





В предыдущем примере:

Для вычисления последнего выражения компилятор выполняет примерно следующие действия:

- сначала переменная `C` целочисленного типа `char` явно приводится к целочисленному типу `int` путём расширения разрядности;
- выполняется вычисление операндов для операции умножения. Левый операнд имеет тип `int`. Правый операнд — константа `10`, а такие константы по умолчанию имеют тип `int`. Так как оба операнда оператора «`*`» имеют тип `int`, неявное приведение типов не выполняется. Результат умножения тоже имеет тип `int`;
- выполняется вычисление операндов операции сложения. Левый операнд — результат умножения имеет тип `int`. Правый операнд — переменная `Y` имеет тип `int`. Так как оба операнда оператора «`+`» имеют тип `int`, неявное приведение к общему типу не выполняется. Результат сложения тоже имеет тип `int`;
- выполнение присваивания. Левый операнд — переменная `X` имеет тип `int`. Правый операнд — результат вычисления выражения, записанного вправа от знака «`=`», тоже имеет тип `int`. Так как оба операнда оператора «`=`» имеют одинаковый тип, неявное приведение типов не выполняется.



Если в выражении появляются операнды различных типов, то они преобразуются к некоторому общему типу, при этом к каждому арифметическому операнду применяется такая последовательность правил:

- Если один из операндов в выражении имеет тип `long double`, то остальные тоже преобразуются к типу `long double`.
- В противном случае, если один из операндов в выражении имеет тип `double`, то остальные тоже преобразуются к типу `double`.
- В противном случае, если один из операндов в выражении имеет тип `float`, то остальные тоже преобразуются к типу `float`.
- В противном случае, если один из операндов в выражении имеет тип `unsigned long`, то остальные тоже преобразуются к типу `unsigned long`.



Если в выражении появляются операнды различных типов, то они преобразуются к некоторому общему типу, при этом к каждому арифметическому операнду применяется такая последовательность правил:

- В противном случае, если один из операндов в выражении имеет тип `long`, то остальные тоже преобразуются к типу `long`.
- В противном случае, если один из операндов в выражении имеет тип `unsigned`, то остальные тоже преобразуются к типу `unsigned`.
- В противном случае все операнды преобразуются к типу `int`. При этом тип `char` преобразуется в `int` со знаком; тип `unsigned char` в `int`, у которого старший байт всегда нулевой; тип `signed char` в `int`, у которого в знаковый разряд передается знак из `char`; тип `short` в `int` (знаковый или беззнаковый).



В операциях присваивания тип значения, которое присваивается, преобразуется к типу переменной, получающей это значение. Допускается преобразования целых и плавающих типов, даже если такое преобразование ведет к потере информации.



Преобразование целых типов со знаком. Целое со знаком преобразуется к более короткому целому со знаком, посредством усечения старших битов. Целая со знаком преобразуется к более длинному целому со знаком, путем размножения знака. При преобразовании целого со знаком к целому без знака, целое со знаком преобразуется к размеру целого без знака и результат рассматривается как значение без знака.



Если в выражении появляются операнды различных типов, то они преобразуются к некоторому общему типу, при этом к каждому арифметическому операнду применяется такая последовательность правил:

- В противном случае, если один из операндов в выражении имеет тип `long`, то остальные тоже преобразуются к типу `long`.
- В противном случае, если один из операндов в выражении имеет тип `unsigned`, то остальные тоже преобразуются к типу `unsigned`.
- В противном случае все операнды преобразуются к типу `int`. При этом тип `char` преобразуется в `int` со знаком; тип `unsigned char` в `int`, у которого старший байт всегда нулевой; тип `signed char` в `int`, у которого в знаковый разряд передается знак из `char`; тип `short` в `int` (знаковый или беззнаковый).



Циклы используются для многократного повторения участков кода. Возможность повторения определенных фрагментов кода — это одна из основных и в тоже время важных задач, которые приходится решать программисту.



Большинство программ или сайтов используют циклы, например — для вывода новостной информации или объявлений. То есть в таких задачах необходимо выполнять постоянно операции чтения и записи, и для того чтобы не дублировать один и тот же код на помощь приходят циклы.

Циклы достаточно просто объявляются в коде, однако они выполняют сложные задачи, всего лишь простым повторением.





В языке Си существует три типа циклов:

- for
- while
- do while



Самый часто используемый цикл — это цикл for, его структура показана ниже:

```
for ( инициализация переменной; условие; изменение  
значения переменной )  
{  
  
    // тело цикла  
  
}
```



```
for ( инициализация переменной; условие; изменение  
      значения переменной )  
{  
    // тело цикла  
}
```

Инициализация переменной позволяет либо объявить переменную и присвоить ей значение либо присвоить значение уже существующей переменной.

Во-вторых, значение этой переменной сообщает программе — истинно или ложно условие цикла.

И пока условие цикла — истинно, цикл должен продолжать повторяться.



Каждую секцию в заголовке цикла, отделяет точка с запятой ,  
что очень важно.



Также отметим, что каждый из разделов может быть пустым,  
хотя точки с запятой все еще должны быть там.

Если условие не пустое, то оно оценивается как истинное и  
цикл будет выполняться до тех пор, пока что-то не сделает  
условие цикла — ложным.



## Пример

```
int i;  
  
for ( i = 0; i < 10; i++ ) {  
  
    printf( "%d\n", i );  
}  
getchar();
```

Результат:

```
0  
1  
2  
3  
4  
5  
6
```



## Пример

```
int num;  
for (num=10; num<20; num=num+1)  
{  
    printf("%d\n",num);  
}
```

Результат:

```
10  
11  
12  
13  
14  
15  
16
```



## Пример

```
int num=10;  
for (;num<20;num++)  
{  
    printf("%d\n",num);  
}
```

Результат:

```
10  
11  
12  
13  
14  
15  
16
```



## Пример

```
int num=10;  
for (num=10; num<20; )  
{  
    num++;  
    printf("%d\n",num);  
}
```

Результат:

```
11  
12  
13  
14  
15  
16  
17
```





## Пример

```
int num=10;  
for (;num<20;  
{  
  
    num++;  
    printf("%d ",num);  
}
```

Результат:

```
11 12 13 14 15 16 17 18 19 20
```



## Пример

```
for (int i=0; i<=10; i++)  
{  
    for (int j=0; j<=10; j++)  
    {  
        printf("[%d,%d]",i ,j);  
    }  
}
```

Результат:

```
[0,0][0,1][0,2][0,3][0,4][0,5][0,6][0,7][0,8][0,9][0,10][1,0][1,1][1,2][1,3][1,4][  
1,5][1,6][1,7][1,8][1,9][1,10][2,0][2,1][2,2][2,3][2,4][2,5][2,6][2,7][2,8][2,9][  
2,10][3,0][3,1][3,2][3,3][3,4][3,5][3,6][3,7][3,8][3,9][3,10][4,0][4,1][4,2][4,3][  
4,4][4,5][4,6][4,7][4,8][4,9][4,10][5,0][5,1][5,2][5,3][5,4][5,5][5,6][5,7][5,8][  
5,9][5,10][6,0][6,1][6,2][6,3][6,4][6,5][6,6][6,7][6,8][6,9][6,10][7,0][7,1][7,2][  
7,3][7,4][7,5][7,6][7,7][7,8][7,9][7,10][8,0][8,1][8,2][8,3][8,4][8,5][8,6][8,7][  
8,8][8,9][8,10][9,0][9,1][9,2][9,3][9,4][9,5][9,6][9,7][9,8][9,9][9,10][10,0][10,  
,1][10,2][10,3][10,4][10,5][10,6][10,7][10,8][10,9][10,10]
```



Тело цикла начинает выполняться, если условие цикла — истинно.

Условие представляет собой логическое выражение, например  $x == 1$  или  $x \neq 7$  ( $x$  не равно 7).

То есть условие может быть абсолютно любым — любое сочетание логических выражений.

```
while ( /*условие*/ )  
{  
    // тело цикл - тут находится код, который необходимо повторять  
}
```



Достаточно просто использовать данный цикл:

```
#include <stdio.h>

int main()
{
    int var = 0;

    while ( var < 10 )
        printf( "%d", var );
        var++;
    }
    getchar();
}
```

Результат:

```
1 2 3 4 5 6 7 8 9
```



Этот цикл полезен, когда необходимо выполнить код по крайней мере — 1 раз.

Рассмотрим его структуру:

```
do {  
    // тело цикла  
} while ( /*условие*/ );
```



Достаточно просто использовать данный цикл:

```
#include <stdio.h>

int main()
{
    int i = 0;

    do {
        /* Напечатает сообщение и завершит работу*/
        printf( "Привет! Я цикл do while\n" );
    } while ( i != 0 );
    getchar();
}
```

Результат:

```
Привет! Я цикл do while
```



Во всех указанных операторах можно использовать:

`continue` – когда необходимо пропустить код, следующий за `continue` и перейти к следующей итерации

`break` – когда необходимо досрочно прекратить выполнение цикла

```
#include<stdio.h>
int main()
{
    int i;
    i = 0;
    while ( i < 20 )
    {
        i++;
        if ( i == 10)
            break;
    }
    return 0;
}
```



Во всех указанных операторах можно использовать:

`continue` – когда необходимо пропустить код, следующий за `continue` и перейти к следующей итерации

`break` – когда необходимо досрочно прекратить выполнение цикла

```
#include<stdio.h>
int main()
{
    int i;
    i = 0;
    while ( i < 20 )
    {
        i++;
        continue;
        printf("Nothing to see\n");
    }
    return 0;
}
```





Для открытия файла необходимо использовать функцию:

```
FILE *fopen(const char *filename, const char *mode);
```

При этом доступны режимы:

r - открыть для чтения

w - открыть для записи (файл может не существовать)

a - открыть для добавления (файл может не существовать)

r+ - открыть для чтения и записи, установив позицию вначале файла

w+ - открыть для чтения и записи (перезаписать файл)

a+ - открыть для чтения и записи (добавляется, если файл существует)



Простой пример открытия файла:

```
FILE *fp;  
fp=fopen("c:\\test.txt", "r");
```

В данном случае открывается файл test.txt для чтения в текстовом виде. Чтобы считать этот файл в бинарном виде необходимо добавить символ:

«b»

к символу «r», т.е.

```
fp=fopen("c:\\test.txt", "rb");
```



После работы с файлом его необходимо закрыть:

```
int fclose(FILE *a_file);
```

Пример;

```
FILE *fp;  
fp=fopen("c:\\test.txt", "r");  
fclose(fp);
```



Запись в файл производится функцией:

```
fprintf(FILE *a_file, const char *format, args);
```

Пример;

```
#include<stdio.h>
int main()
{
    FILE *ptr_file;
    int x;
    ptr_file =fopen("output.txt", "w");
    if (!ptr_file)
        return 1;
    for (x=1; x<=10; x++)
        fprintf(ptr_file,"%d\n", x);
    fclose(ptr_file);
    return 0;
}
```



Чтение из файла производится функциями:

```
fgets(char *buf, int length, FILE *ptr_file);  
fscanf (FILE *a_file, const char *format,args);
```

Функция fgets читает файл по строкам, заданной длины (при этом символ \n так же считывается)

Функция fscanf аналогична функции scanf и выполняет форматированное считывание.



Пример:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    char str1[10], str2[10], str3[10];
    int year;
    FILE * fp;
    fp = fopen ("file.txt", "w+");
    fputs("We are in 2012", fp);
    rewind(fp);
    fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
    printf("Read String1 |%s|\n", str1 );
    printf("Read String2 |%s|\n", str2 );
    printf("Read String3 |%s|\n", str3 );
    printf("Read Integer |%d|\n", year );
    fclose(fp);
    return(0);
}
```

```
Read String1 |We|
Read String2 |are|
Read String3 |in|
Read Integer |2012|
```



Пример:

```
#include<stdio.h>
int main()
{
FILE *ptr_file;
char buf[1000];
ptr_file =fopen("input.txt","r");
if (!ptr_file)
    return 1;
while (fgets(buf,1000, ptr_file)!=NULL)
    printf("%s",buf);    fclose(ptr_file);
return 0;
}
```

Файл открывается для чтения, функция fgets вернет NULL, если будет достигнут конец файла. Каждая строка выводится на консоль.



Массивы в языке си объявляются достаточно просто:

```
int num[35]; /* Массив целых чисел размером в 35 элементов */  
char ch[10]; /* Массив символов из 10 элементов*/
```





## Пример

```
#include <stdio.h>
int main()
{
    int avg = 0;
    int sum = 0;
    int x=0;
    int num[20];
    for (x=0; x<=19;x++)
    {
        num[x]=x+1;
    }
    for (x=0; x<=19;x++)
    {
        sum = sum+num[x];
    }
    avg = sum/20;
    printf("%d", avg);
    return 0;
}
```

Результат:  
10



Способ объявления массивов может быть таким:

```
int arr1[5] = {1, 2, 3, 4 ,5};
```

```
int arr1[] = {1, 2, 3, 4, 5};
```



Массивы могут быть двумерными:

```
#include<stdio.h>
int main()
{
    int disp[3][5];
    int i, j;
    for(i=0; i<=2; i++)
    {
        for(j=0; j<=4; j++)
        {
            disp[i][j]=i+j;
        }
    }
    for(i=0; i<=2; i++)
    {
        for(j=0; j<=4; j++)
        {
            printf("%d", disp[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



## Результат

01234

12345

23456



## Пример объявления и инициализации двумерного массива

```
int disp[2][4] = {  
    {10, 11, 12, 13},  
    {14, 15, 16, 17}  
};
```

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

```
int abc[2][2] = {1, 2, 3 ,4 }  
int abc[][2] = {1, 2, 3 ,4 }
```



Спасибо за внимание !

