

Перевод натурального числа из десятичной системы счисления в двоичную.

```
#include <iostream>
using namespace std;
void DecToBin( int n ) {
    if ( n >= 2 ) {
        DecToBin( n/2 );
    }
    cout << n % 2;
    return;
}

int main () {
    int n;
    cout << "n = ";
    cin >> n;
    cout << n << " (Dec) = ";
    DecToBin( n );
    cout << " (Bin)" << endl;
    return 0;
}
```

n = 65 (Dec) = 100001 (Bin)



**Понятие
указателя.
Операции с
указателями.**

План лекции:

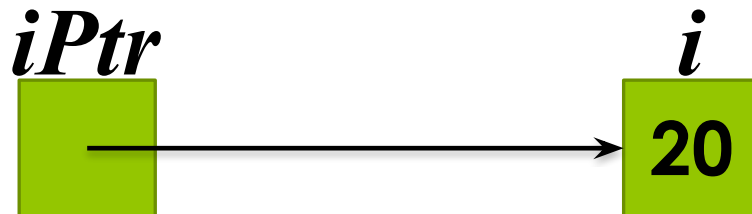
1. Понятие указателя, виды указателей
2. Объявление и разыменовывание указателей
3. Операции с указателями
4. Нулевые указатели
5. Указатели и массивы

Указатель — элемент программы, хранящий адреса памяти некоторого объекта (например, переменной) определённого типа.

Указатель — это переменная, которая содержит адрес некоторого объекта.

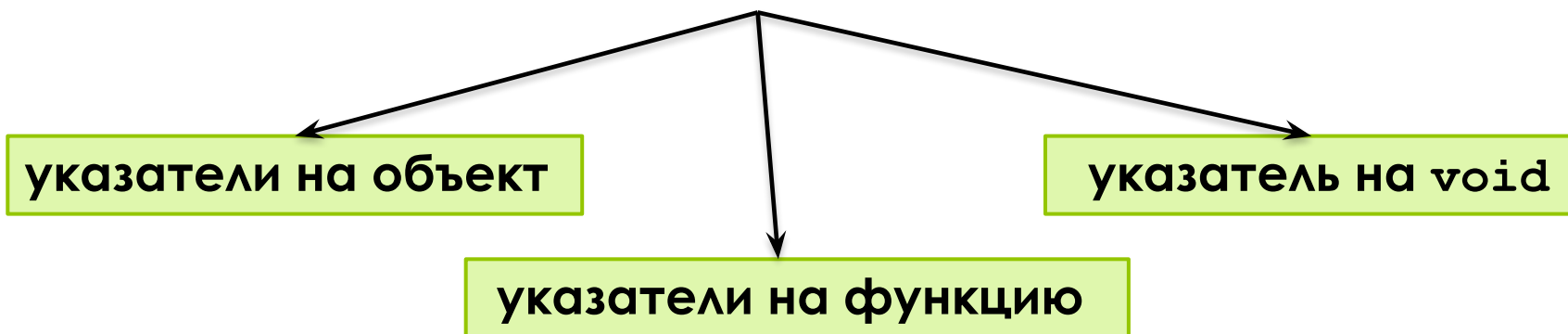


i прямо ссылается на переменную со значением 20



iPtr косвенно ссылается на переменную со значением 20

Виды указателей:



Указатель на функцию – адрес в сегменте кода, по которому передается управление вызова функции.

тип (*имя) (список_типов_аргументов) ;

Пример:

```
int (*fun) (double, double) ;
```

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного).

тип *имя;

Пример:

```
int *a, b, *c;
```

Указатель на void применяют, когда конкретный тип объекта, адрес которого надо хранить, не определен.

```
void *pv;  
float f, *pf;  
pf = &f;  
pv = pf;  
pf = (float*)pv;
```

Указатель может быть константой или переменной, а также указывать на константу или переменную.

Примеры:

```
int pi;  
const int ci=1;  
int *pi;  
const int *pci;  
int *const cpi=&i;  
const int *const cpc=&ci;
```

2. Объявление и разыменовывание указателей

```
int *countPtr; //countPtr указывает на объект типа int
```

```
double *xPtr, *yPtr, z;
```

Главная операция над указателями – это *косвенное обращение* – **разыменовывание** – обращение к объекту, на который настроен указатель.

```
char c1 = 'a';
```

```
char* p = &c1;
```

```
char c2 = *p;
```


3. Операции с указателями

- & — взятие адреса

```
p=&v[3];
```

Операция взятия адреса является унарной операцией, возвращающей адрес в памяти своего операнда.

- * — возврат значения по указанному адресу

```
int x=10;  
int *p, *g;  
p = &x;  
g = p;  
cout << *g;
```

- **$p+n$** , где **p** — указатель, **n** — целое положительное число. Результат — некоторый указатель, полученный смещением p на n позиций вправо.
- **$p-n$** , где **p** — указатель, **n** — целое положительное число. Результат — некоторый указатель, полученный смещением p на n позиций влево.
- **$p-q$** , где **p** и **q** — указатели на один и тот же тип. Результат — целое число, равное количеству шагов, на которое нужно сместить q вправо, чтобы он достиг указателя p , также этот результат можно называть “расстоянием” между указателями, оно может быть и отрицательным, если элемент, на который направлен указатель q расположен правее (то есть, далее), чем элемент, на который направлен указатель p .
- **$p++$ (инкремент)**, **$p--$ (декремент)**, где p — указатель. $p=p+1$, $p=p-1$ соответственно.

Константный указатель нельзя перемещать (записывать в него другой адрес), но можно его разыменовывать или делать его участников вышеперечисленных операций:

```
int ar[] = {-72, 3, 402, -1, 55, 132};  
cout << *ar;  
int* p = ar+3;  
p--;  
cout << *p;
```

`a[i] == *(a+i),`

где **a** указатель на массив любого типа и **i** допустимый индекс этого массива

Функция, подсчитывающая число символов в строке:

```
int strlen(char* p)
{
    int i = 0;
    while (*p++) i++;
    return i;
}
```

```
int strlen(char* p)
{
    char* q = p;
    while (*q++) ;
    return q-p-1;
}
```

4. Нулевые указатели.

Указатель со значением **0** или **NULL** ни на что не указывает и называется *нулевым указателем*.

Если указатель определён одним типом, а переменная, на которую он будет указывать другого типа то необходимо, перед присваиванием адреса, обязательно выполнять преобразование типов.

```
float x;  
int *p;  
p = (int*)&x;
```

5. Массивы и указатели

```
int b[5]; //создать 5-элементный целый массив b  
int *bPtr; // создать указатель bPtr на целое
```

Присвоение указателю адрес первого элемента массива b:

```
bPtr=b;
```

```
bPtr=&b[0];
```

```
*(bPtr+3)
```

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

int main()
{
int b[]={10, 20, 30, 40};
int *bPtr=b;
    cout<<"Massiv b:"<<endl;
for (int i=0; i<4; i++)
    cout<<"b["<<i<<"]="<<b[i]<<endl;

//вывести массив b, используя bPtr и смещение
    cout<<"\n Notacia Pointer/Offset:\n";
for (int offset=0; offset<4; offset++)
    cout<<"*(bPtr + "<<offset<<")= "<<*(bPtr +
offset)<<' \n' ;

return 0;
}
```

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "Rus");
    int array[8]={1,2,3,4,5,6,7,8};
    int *Ptr;
    for (int i=1; i<8; i+=2)
    { cout<<"Элемент массива с индексом ["<<i<<"] имеет
значение "<<*(array+i)<<endl;}
    system("pause");
    return 0;
}
```

```
Элемент массива с индексом [1] имеет значение 2
Элемент массива с индексом [3] имеет значение 4
Элемент массива с индексом [5] имеет значение 6
Элемент массива с индексом [7] имеет значение 8
Для продолжения нажмите любую клавишу . . .
```




Спасибо за внимание!

Дисциплина: Конструирование программ и языки программирования

Тема: Понятие указателя. Операции с указателями.

Преподаватель: Гайшун Алеся Александровна ©