Указатели и ссылки

Определения

Указатель (pointer) – это переменная, значением которой является адрес другой переменной.
Тип указателя обязательно должен совпадать с типом переменной, адрес которой он хранит.

Применение указателей предоставляет возможность создавать динамические структуры данных, размер которых определяется не при компиляции программы, а в процессе ее исполнения.

Тип* Идентификатор ;
Тип *Идентификатор ;
Тип *Идентификатор1, …, *ИдентификаторN ;

```
char* ps ;
float *ptr ;
int *px, *py ;
char* p, ch ;
char* ps2 (0) ;
int *px2 = 0;
```

Нельзя использовать в программе указатель, значение которого не определено.

Операции с указателями

Название	Знак	Пояснения
Присваивание	=	Присвоить указателю адрес переменной или 0.
Инкремент	++	Увеличить указатель на 1. После увеличения указатель будет указывать на следующий элемент массива.
Декремент	--	Уменьшить указатель на 1. После уменьшения указатель будет указывать на предыдущий элемент массива.
Сложение	+	Увеличить указатель на целое значение.
Сложение с замещением	+=	Увеличить существующий указатель на целое значение.
Вычитание	–	Уменьшить указатель на целое значение или вычесть другой указатель, если оба указателя указывают на элементы одного и того же массива.
Вычитание с замещением	–=	Уменьшить существующий указатель на целое значение или вычесть другой указатель, если оба указателя указывают на элементы одного и того же массива.
Отношения	== != > < >= <=	Сравнить указатели. Возвращается истина, если операция сравнения закончилась успешно, иначе возвращается ложь. Сравниваемые указатели должны быть каким-то образом связаны, иначе операция не имеет смысла.

Операции с указателями

Арифметические операции, связанные с увеличением указателя на целое значение, модифицируют значение указателя на число, соответствующее произведению этого числа на количество занимаемых типом указателя байтов. Инкремент модифицирует указатель не на 1, а на число байтов, которое занимает тип указателя. Арифметические операции, которые уменьшают указатель на целое число, выполняют такую же модификацию, но в сторону уменьшения значения адреса.

Специальные операторы с указателями

Название	Знак	Пояснения
Взятие адреса	**&**	Получить адрес переменной.
Разыменовывание	*	Получить значение переменной по её адресу.
Выделение памяти	**new**	Получить указатель на начало выделенного блока оперативной памяти.
Освобождение памяти	**delete**	Освободить выделенный блок памяти и сделать указатель недоступным.

Операции с указателями

```
int x (5) ;
int * px (& x) ; // взятие адреса x

// вывод адреса x (0012FF60)
cout << "Address x\t" << & x << endl ;

// вывод значения указателя (0012FF60)
cout << "Pointer px\t" << px << endl ;

// объявление переменной и указателя
double v (0.05), *pv (0) ;

// взятие адреса v
pv = & v ;
double x (*pv) ;
cout << "x = " << x << endl ; // x = 0.05
cout << *pv << endl ; // вывод значения v через pv 0.05

// int p = &10;
// doubple pd = &(7 + v / 200);
```

Операторы new и delete

Оператор **new** выделяет блок памяти и возвращает указатель на первую ячейку этого блока. Если new не в состоянии найти необходимое пространство свободной памяти, он возвращает указатель 0.

```
ИмяУказателя = new Тип ;
ИмяУказателя = new Тип (Значение) ;
Тип* ИмяУказателя (new Тип) ;
Тип* ИмяУказателя (new Тип (Значение)) ;
```

```
int* pi ; unsigned short* pu ;

pi = new int ;

// cout << pu << endl;
pu = new unsigned short ( 200 ) ;
cout << pu << endl;

double* pd (new double) ;
float* pf (new float (-3.15)) ;
```

Операторы new и delete

Оператор **delete** освобождает память, выделенную ранее оператором new. Недо-пустимо применение оператора delete для указателя, который не участвовал в выполнении оператора new.

```
delete ИмяУказателя ;
```

```
int x (25) ;
int* py (new int ( x )) ;
*py += 5;

cout << *py << endl ;
cout << x << endl;

delete py ;
// delete py ;

int *pz = &x;
*pz += 5;
cout << *pz << endl;
cout << x << endl;
// delete pz
```

Указатели и массивы

```
int array [10];
```

Имя массива array является его константным адресом, поэтому выражение (array+i) представляет адрес i-го элемента в массиве array.

Чтобы получить значение по этому адресу, необходимо выполнить операцию разыменовывания, то есть записать выражение *(array+i).

Для получения адреса i-го элемента массива следует использовать операцию взятия адреса &array [i].

```
int array [ ] = { 10, 20, 30, 40, 50 } ;
int size = sizeof (array) / sizeof (array [0]) ;
int* pTop (array) ;          // int* pTop (&array [0]) ;
int* pEnd (array+size) ;     // int* pEnd (&array [size]) ;
int* p (pTop) ;
while (p < pEnd) {
    cout << *p << '\t' ;
    p++ ;
}
cout << endl ;
```

Указатели и массивы

```
char s [ 20 ] ;
char* p (s) ;
cout << "-> " ;
cin >> p ;
while (* p){
   cout << p++ << endl ;
}

p = s ;
cout << "\nInitial string -> " << p << endl ;
```

Указатели и динамические массивы

Операторы new и delete позволяют выделять и освобождать память при работе с динамическими массивами, размер которых становится известным в процессе выполнения программы.

```
Тип* ИмяУказателя (new Тип [Размер]) ;
ИмяУказателя = new Тип [Размер] ;
delete [ ] ИмяУказателя ;
```

```
int size ; cin >> size ; // ввод размера массива

// выделение памяти под массив
double* pd (new double [size]) ;

// выделение памяти под массив
int* pi ; pi = new int [size] ;

delete [ ] pi ;
delete [ ] pd ; // освобождение памяти
```

Указатели и динамические массивы

```
unsigned char size = 0;
std::cin >> size;
if (size > 2)
{
    //int array[size];
    int *p = new int[size];

    int* pA = p;
    int* pB = pA + 1;
    *pA = 10;
    std::cout << p[0] << std::endl;

    *pB = 20;
    std::cout << p[1] << std::endl;

    std::cout << pB - pA << std::endl;       // 1
    std::cout << (int)pB - (int)pA << std::endl;//sizeof(int)

    pB++;
    *pB = 30;
    std::cout << p[2] << std::endl;
}
```

void*

```
int a = 10;
int *pA = &a;
//float *pF = &a;

void *pV = &a;
//std::cout << *pV << std::endl;

//int* pB = pV;
int* pC = (int*)pV;

*pC = 20;
```

Указатель на указатель

Тип Идентификатор ;**

```
// ввод количества строк и стролбцов
int row, col ; cin >> row ; cin >> col ;
int** p; // объявление указателя на указатель
// выделение памяти под адреса строк матрицы
p = new int* [row] ;

// выделение памяти под строки матрицы
for (int j (0); j < row; j++)
   p [j] = new int [col] ;

for (int i (0); i < row; i++)
   for (int j(0); j < col; j++)
     cin >> p [i][j] ;

// освобождение памяти, занятой строками
for (int j(0); j < row; j++)
   delete [ ] p [j] ;

// освобождение памяти, занятой адресами строк
delete [ ] p ;
```

Указатель и const

Указатель на константу:

const Тип* ИмяУказателя ;
const Тип* ИмяУказателя (адрес константы) ;

```
const char A = '*' ;
const char B = '!' ;
char C = ':';

const char* pC (&A) ;
pC = &B;

// *pC = '.' ;

pC = &C;
// *pC = '.';
```

Указатель и const

Константный указатель:

Тип* const ИмяУказателя (адрес) ;

```
char a = '*' ;
char b = '!';
const char c = ':';

char* const pA (&a);
// pA = &b;

*pA = '.';

// char* const pC (&c) ;
```

Ссылки

Ссылка (reference) – является альтернативным именем переменной, указанной при инициализации ссылки. Ссылка является переменной, которая содержит адрес другой переменной. По существу – это неявный указатель с константным значением адреса.

Особенности ссылок:

1. Ссылка при объявлении **обязательно должна быть проинициализирована (исключением является объявление ссылки в качестве члена-данного).**
2. Значение ссылки **не может быть изменено в ходе работы программы.**
3. Для получения данных по ссылке **не надо пользоваться операцией разыменовывания.**
4. **Нельзя создавать указатель на ссылку (у ссылки нет адреса).**
5. **Нельзя создавать массивы ссылок.**

```
Тип& ИмяСсылки (ИмяЯвнойПеременной) ;
```

Ссылки

```
int x = 10;
int& rX (x) ;

std::cout << rX << std::endl;

rX += 10;
std::cout << x << std::endl;

double& rY (*new double);
int i (5) ;
int& r (i) ;
int* p1 (&i) ;
int*& s = p1 ;

//double& rY ;
```

Ссылки

```
int x = 10;
int& rX = x;
int y = rX;

rX = 20;

std::cout << x << std::endl;
std::cout << y << std::endl;
std::cout << rX << std::endl;

const int& crX = x;
// crX ++;

// int& r = x + 10;
```