Уральский федеральный университет

Указатели

Лекция 10

Иллюстративный материал к лекциям по алгоритмизации и программированию

Автор Саблина Н.Г. 2016 г.

Оглавление

Описание указателя

Указатели - переменные для хранения адресов областей памяти.

В C++ различают три вида указателей:

- на объект;
- на функцию;
- на тип *void*.

Указатель на функцию

Содержит адрес в сегменте кода, по которому располагается исполняемый код функции.

Указатель функции имеет тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа»:

тип (*имя) (список типов аргументов);

Указатель на объект

Содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного).

Объявление указателя на объект :

```
тип *имя;
```

Указатель на **void**

Применяется, когда тип объекта не определен.

Указателю на **void** можно

- присвоить значение указателя ***любого*** типа,
- сравнивать его с любыми указателями,
- но ***перед выполнением*** каких-либо действий с областью памяти, на которую он ссылается, ***нужно преобразовать*** его к конкретному типу явным образом.

Использование указателей (1 из 3)

- при работе с динамической памятью, называемой кучей (heap).

Куча - *свободная память, в которой можно во время выполнения программы выделять место в соответствии с потребностями программы*.

- Доступ к выделенным участкам динамической памяти производится только через указатели.
- Время жизни динамических переменных - от создания до конца программы или до явного освобождения памяти.

Использование указателей (2 из 3)

- Перед использованием указателя его надо инициализировать, т. е. присвоить значение.
- Использование неинициализированных указателей - источник ошибок в программах.
- Инициализатор записывается после имени указателя либо в круглых скобках, либо после знака равенства.

Инициализация указателей (1 из 3)

Способы инициализации указателя:

- Присваивание указателю адреса существующего объекта
- Присваивание указателю адреса области памяти в явном виде
- Присваивание пустого значения
- Выделение участка памяти

Присваивание указателю адреса существующего объекта

- с помощью операции получения адреса:

```
int a=5;          // целая переменная
int* p=&a         // в указатель записывается адрес a
int* p (&a);      // то же самое другим способом
```

- с помощью значения другого инициализированного указателя:

```
int* r = p:
```

- помощью имени массива или функции, которые трактуются как адрес:

```
int B[10]:
int* t=B;         // присваивание адреса начала массива
void f(int a ) {(/* ... */ }          // определение функции
void (*pf)(int);  // указатель на функцию
pf=f;     // присваивание адреса функции
```

Присваивание указателю адреса области памяти в явном виде

char* vp = (char *) 0xB8000000;

- Здесь
  - 0xB8000000 - шестнадцатеричная константа,
  - (cha r *) - операция приведения типа:
  - константа преобразуется к типу «указатель на char».

Присваивание указателю адреса области памяти в явном виде (доп)

Пример.

char * const key_byte =(char *)1047;

- Значение указателя невозможно изменить, он всегда показывает на байт с адресом 1047 (шестнадцатеричное представление 0x1047).
- Это - байт состояния клавиатуры. Содержимое этого участка с помощью разыменования указателя-константы как для чтения, так и для изменений.

```
#include <iostream.h>
void main()
{char *const key_b=(char *)1047;
cout<<"\n Байт состояния клавиатуры - "<< *key_b;
*key_b='Ё';
cout<<"\n Байт состояния клавиатуры - "<< *key_b;
}
```

Присваивание указателю пустого значения

- Можно использовать константу NULL, определенную как указатель, равный нулю,
- Можно использовать и просто 0

```
int* sx = NULL
int* rz = 0;
```

Выделение участка памяти (1)

- А) с помощью операции **new**, синтаксис:

new имя_типа (инициализатор)

- Б) с помощью функций **malloc**() и **calloc**(). Синтаксис:

имя_указателя=(тип_указателя)malloc(объем_в_байтах);

имя_указателя=(тип_указателя)calloc (число_элементов, размер_эл-та_в_байтах);

Освобождение динамично выделенной памяти ***delete***, ***free***.

Выделение участка памяти (2)

- Примеры:

```
int* n = new int;          //1
int* b = new int (10);     //2
int* q = new int [10];     //3

int* u = (int *) malloc (sizeof (int));        //4
int* u = (int *) calloc (1, sizeof (int));     //5
```

Пример: ***delete*** *n;* ***delete*** *[ ]b;*
free *(u);*

Действия с указателями

С указателями можно выполнять следующие операции:

1) разыменование, или косвенное обращение к объекту (*),
2) присваивание,
3) приведение типов,
4) сложение с константой,
5) вычитание,
6) инкремент (++),
7) декремент (--),
8) сравнение

Операции: разадресации, или разыменования

Разыменование - доступ к величине, адрес которой хранится в указателе.

Можно использовать как для получения значения, так и для изменения значения:

```
char   a;          // переменная типа char
char * p = new char;    /* выделение памяти под указатель и под
    динамическую переменную типа char */
*p = 'Ю';    a = *p;       // присваивание значения обеим переменным
```

Присваивание

```
int *t, *z, y=34;
t=&y;   z=t;
```

Приведение типов для указателей (1)

- Пример:

```
#include <iostream.h>
void main()
{ unsigned long L=0x12345678;
  char *cp=(char *)&L;
  int *ip=(int *)&L;
  long *lp=(long *)&L;
  cout<<hex;
  cout<<"\n Адрес L=" <<&L;
  cout <<"\n cp= "<<(void *)cp<<"\t*cp= 0x"<<(int)*cp;
  cout <<"\n ip= "<<(void *)ip<<"\t*ip= 0x"<<*ip;
  cout <<"\n lp= "<<(void *)lp<<"\t*lp= 0x"<<*lp;
}
```

Приведение типов для указателей (2)

На экран возможен следующий вывод:

Адрес L=0x8fe10ffc

cp= 0x8fe10ffc **cp= 0x78*

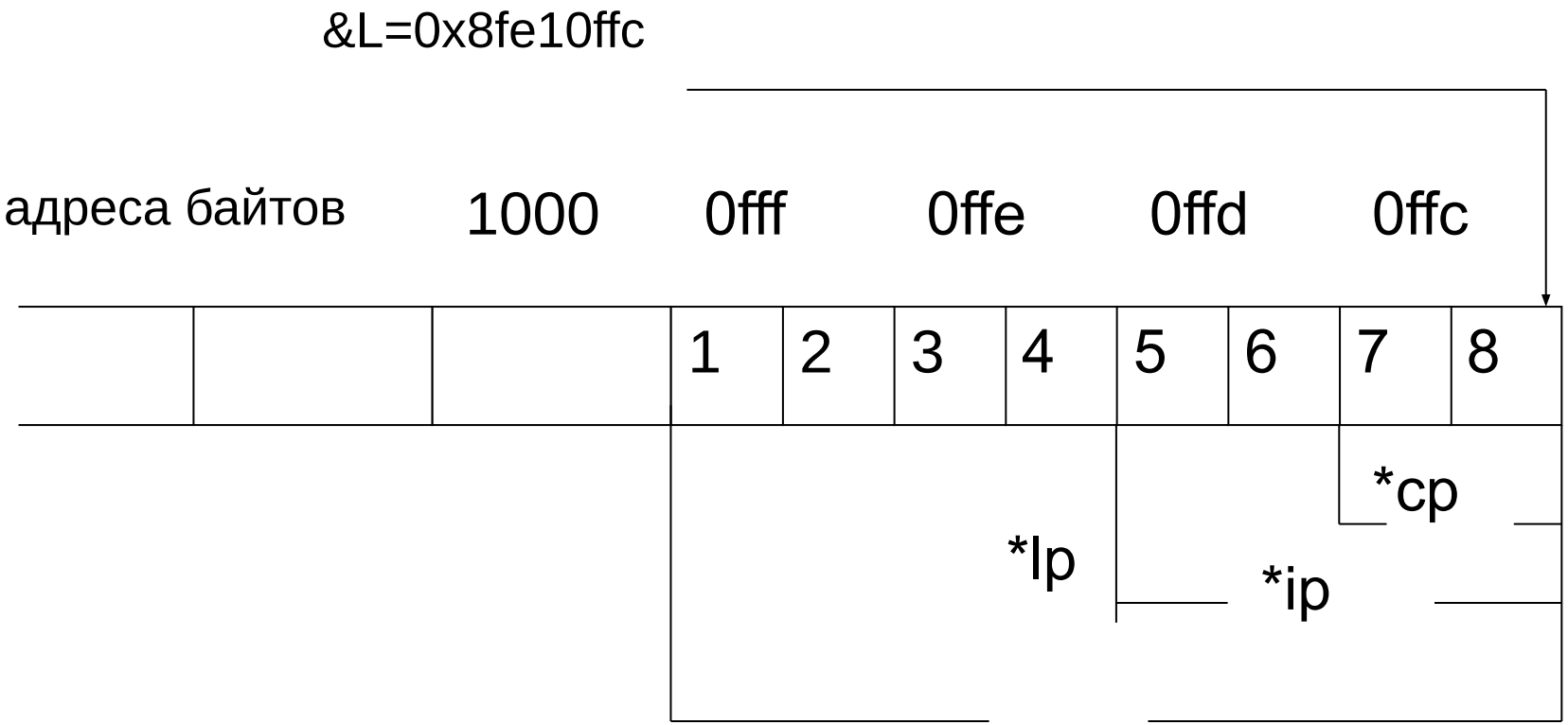
ip= 0x8fe10ffc **ip= 0x5678*

lp= 0x8fe10ffc **lp= 0x12345678*

Присваивание без явного приведения типов допускается в двух случаях:

- указателям типа void*;
- если тип указателей справа и слева от операции присваивания один и тот же.

Схема размещения в памяти переменной L типа unsigned long



Арифметические операции с указателями (1)

- Инкремент

```
char * p = new char [5]:
p++;
// значение p увеличивается на 1
 long * q = new long [53];    q++ ;
// значение q увеличивается на 4
```

- Декремент
- сложение с константой

```
char * p = new char [5]:
int c=3; p=p+c;
// значение p увеличивается на 3
 long * q = new long [53];   q=q+c;
// значение q увеличивается на 12
```

Арифметические операции с указателями (2)

- Вычитание

  Разность двух указателей — это разность их значений, деленная на размер типа в байтах

Суммирование двух указателей **не допускается.**

Связь массивов и указателей

Имя массива – указатель на его начало

```
int A[8]={12, 0, 2, 3, 4, 5, 65, 8, 3};
int *p;
p=A;
p=&A[0];
p=&A;
```

Обращение к отдельным элементам массива

```
int i=2, x=*p;
x=*(p+i);
x=p[i];
```

Динамическое создание массивов

```
#include <iostream.h>

void main()
{ int n; //размерность массива
int *mm;

  cout<<"\n Введите количество элементов массива n=";     cin>> n;
// создание массива
  mm= new int [n];

// заполнение массива
for (int i=0; i<n; i++) mm[i]=i;

// вывод массива на экран
cout<<"\n Массив \n";
for (i=0; i<n; i++) cout<<mm[i] <<"  ";
…….
delete [] mm;          // delete  mm;
}
```

Динамическое создание матрицы (1)

Способ 1. Использование одномерного динамического массива

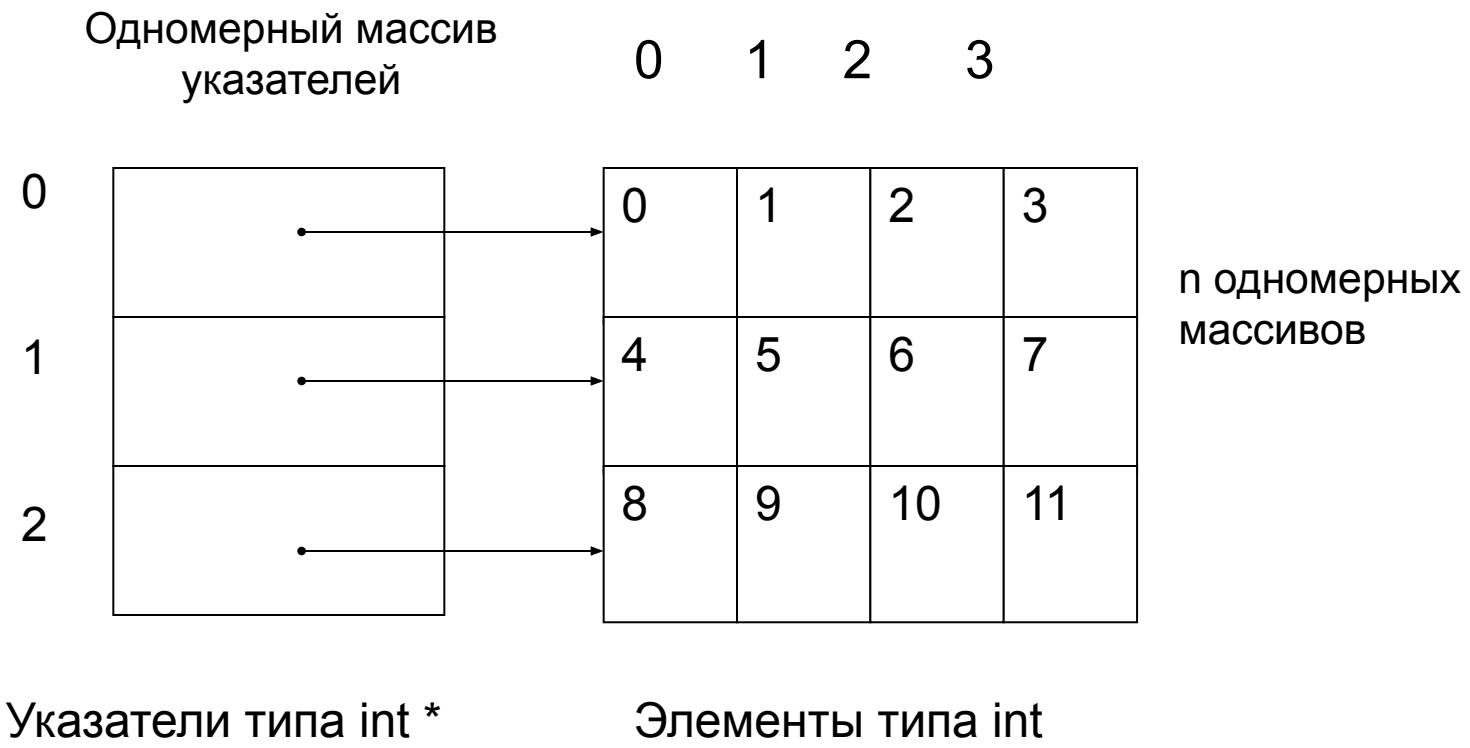
```
int n,m; //размерность  матрицы
  cout<<"\nВведите количество строк матрицы:";     cin>>n;
cout<<"\nВведите количество столбцов матрицы:";  cin>>m;
  int *matr;                  //указатель на начало массива
  matr=new int [n*m];       //выделение памяти для массива
  //заполнение матрицы
for (int i=0; i<n; i++)
       for (int j=0; j<m; j++)
            *(matr+i*m+j)= random(15);
```

Динамическое создание матрицы (2)

Способ 2. Использование вспомогательного динамического массива указателей

Схема.

Схема имитации двумерного массива с помощью массива указателей и набора одномерных массивов



План создания динамической матрицы

1) Описать двойной указатель
2) Задать размерность матрицы
3) Выделить память под вспомогательный массив указателей
4) В цикле выделить память под каждую из строк матрицы (заполнение массива указателей
5) Заполнить матрицу
6) Произвести необходимую обработку
7) Удалить строки матрицы
8) Удалить вспомогательный массив указателей

Динамическое создание матрицы (2)

Способ 2. Использование вспомогательного динамического массива указателей

```
int n,m;                    //размерность  матрицы
cout<<"\nВведите количество строк матрицы:";    cin>>n;
cout<<"\nВведите количество столбцов матрицы:";
    cin>>m;


int **matr;       //указатель на начало массива указателей
//выделение памяти для вспомогательного
//массива указателей
matr=new int *[n];
```

Динамическое создание матрицы (2)

```
//заполнение массива указателей,
//выделение памяти для каждой строки матрицы
for (int i=0; i<n; i++)  matr[i]= new int[m];
//заполнение матрицы
for ( i=0; i<n; i++)
      for (int j=0; j<m; j++)
            matr[i][j]= random(15);
```

Удаление матрицы

```
//обработка элементов матрицы в соответствии
// с условием задачи
….

//удаление строк матрицы
for (i=0; i<n; i++)  delete matr[i];
//удаление вспомогательного массива указателей
delete [] matr;
```

Итоги

Рассмотренные вопросы:

Указатели

- Понятие , способы описания и инициализации
- действия с указателями
- Связь с массивами
- Динамическое выделение памяти под массивы

Библиографический список

- Подбельский В.В. Язык СИ++. Учебное пособие. М.: Финансы и статистика, 2003. – 560 с.
- Павловская Т.А. C/C++. Программирование на языке высокого уровня: учебник для студентов вузов, обучающихся по направлению "Информатика и вычисл. техника" СПб.: Питер, 2005. - 461 с.
- Березин Б.И. Начальный курс C и C++ / Б.И. Березин, С.Б. Березин. - М.: ДИАЛОГ-МИФИ, 2001. - 288 с
- Каширин И.Ю., Новичков В.С. От С к С++. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2005. – 334 с.

Автор:

Саблина Наталья Григорьевна

Ст. преподаватель

каф. РТС УрФУ