Введение в *C++*

Указатели, массивы, строки, функции

Указатели, массивы, строки, функции

Указатели

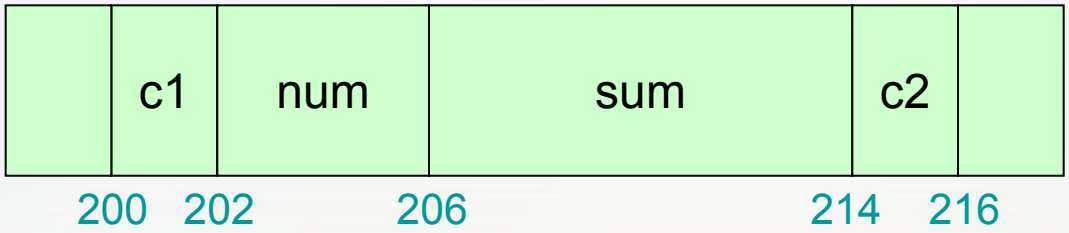
Указатель – это переменная, предназначенная для хранения адреса объекта.

формат: [<тип-объекта>] *<имя-переменной>;

или [<тип-объекта>] *<описатели>; более сложная конструкция

Пусть описаны следующие переменные:

```
char c1;
int num;
double sum;
char c2;
```




```
char *p1;     p1 = &c1;     ( 200 )     c1     ⇔  *p1
int  *p2;     p2 = &num;    ( 202 )     num    ⇔  *p2
double *p3;   p3 = &sum;    ( 206 )     sum    ⇔  *p3
char *p4;     p4 = &c2;     ( 214 )     c2     ⇔  *p4
```

Для эффективной работы с указателями нужно знать размеры объектов, которые они адресуют. Компилятор обычно эти размеры знает (но не всегда).

void *p5; Так тоже можно объявить, но перед использованием указателя нужно явно привести его к некоторому типу, например: (int*) p5++;

Итак: & - операция получения адреса объекта

* - операция косвенной адресации (доступ к объекту через указатель на объект)

Указатели, массивы, строки, функции

Объявления функций, передача аргументов

Сигнатура функции:

[<класс-памяти>] [<тип-результата>] <имя-функции> ([<список-аргументов>]) [throw (<список-объектов-исключений>)];

Сигнатуру ещё называют прототипом, или описанием функции.

Если вместе с прототипом функции также задано тело функции (её исполняемый код), то говорят, что функция определена полностью (дано определение функции).

Список аргументов в прототипе функции является формальным.

Важно лишь количество формальных аргументов и их типы.

Имена формальных аргументов в прототипе функции могут быть опущены.

При вызове функции формальные аргументы замещаются фактическими аргументами.

пример:

```
double f1 ( float, char);          это прототип функции f1
int main ( )  {
int n;  char c;
.....................
f1 ( (float) n, c );               это вызов функции f1
……..……..  }
double f1 ( float x, char y )          это определение функции f1
{  ……..……..}
```

Здесь x и y – формальные аргументы, n и c – фактические аргументы

Указатели, массивы, строки, функции

При вызове функции в специальной области памяти – программном стеке создаются временные копии фактических аргументов. Вызванная функция работает с копиями фактических аргументов. Оригиналы аргументов остаются неизменными. Этот механизм называется передача аргументов «по значению».
Если требуется работать с оригиналами аргументов, имеется механизм передачи аргументов «по ссылке».

пример:

```
interchange (char*, char*);          прототип функции
main()  {
char a =‘a’;
char b=‘b’;
……..
interchange ( &a, &b);          вызов функции
……..
}
interchange (char *p1, char *p2)      определение функции
{
char z;
z = *p1; *p1 = *p2; *p2 = z;
}
```

Указатели, массивы, строки, функции

Ещё примеры функций:

```
char* f2 (void);          функция не имеет аргументов
                          и в качестве результата работы
                          возвращает значение типа char*
                          (указатель на символ)

void f3 (char**, unsigned);       функция имеет два аргумента типа
                          char** и unsigned, никаких значений
                          не возвращает

void* f4 (void*, …);      функция имеет произвольное
                          количество аргументов, но не менее
                          одного. Тип первого аргумента –
                          указатель на любой объект. Тип
                          результата – указатель на любой объект.

void f5 (int x, int &y)   пример явного задания механизма передачи
{                         аргумента «по ссылке»
      x++;                изменяется значение копии аргумента “x”
      y++;                изменяется значение самого аргумента “y”
}
```

Указатели, массивы, строки, функции

Массивы и указатели

Массив есть набор объектов одного типа, размещенных в смежных областях памяти.

Примеры описания массивов:

```
float t[256];     первый элемент t[0], последний элемент t[255]
static char code[12];       - статический массив
extern z[];                 - внешний массив
```

Инициализировать можно только статические и внешние массивы:

```
static a[5] = { 1, 2, 3, 4, 5 };
```

или так:

```
static a[] = { 1, 2, 3, 4, 5 };
```

Компилятор сам определит размерность массива, исходя из списка инициализации.

Пример задания двумерного массива:

```
static b [2][5] = { { 1, 2, 3, 4, 5 }, { 11, 12, 13, 14, 15 } };
```

Вопрос: Какова связь между массивами и указателями?

Ответ: Имя массива есть синоним адреса первого элемента массива. Связь между массивами и указателями задается принципами адресной арифметики.

Адресная арифметика – это приемы манипулирования с указателями.

Указатели, массивы, строки, функции

пример:

Пусть есть следующие описания:

```
int d[4], *p;
p = &d[0];
```

или проще:

```
p = d;
```

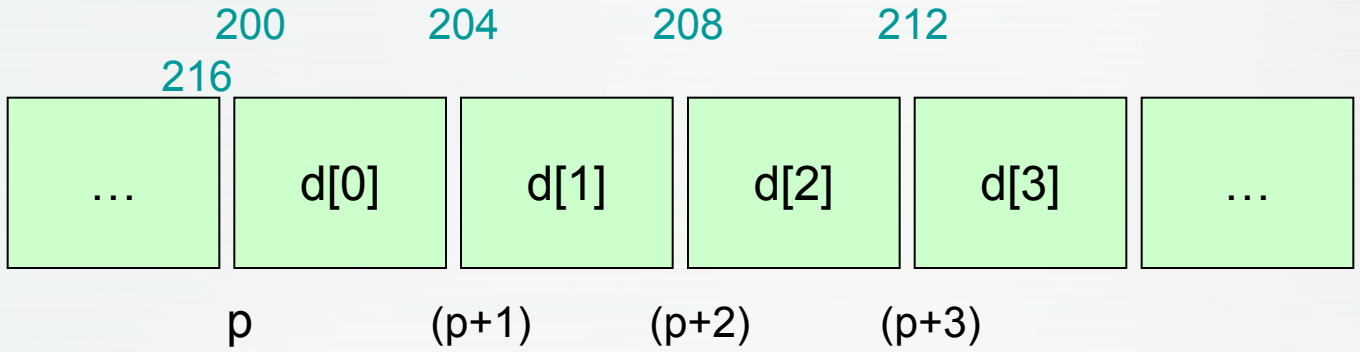
Тогда

*p ⇔ d[0]

*(p+1) ⇔ d[1]

*(p+2) ⇔ d[2]

*(p+3) ⇔ d[3]



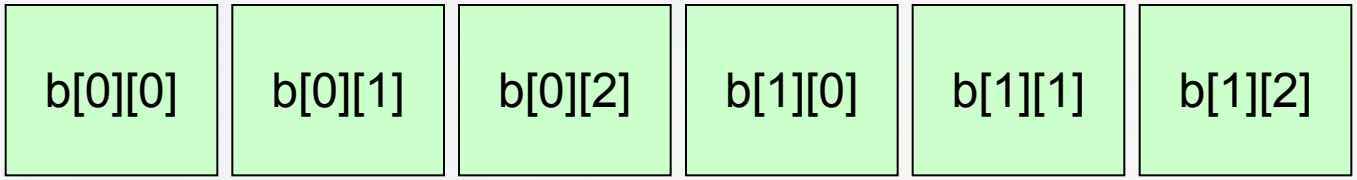
и вообще: *(p+n) ⇔ d[n]

Два принципа адресной арифметики:

1. Пусть p – указатель на объект определенного типа. Тогда p++ указывает на следующий объект в массиве объектов данного типа, а p-- указывает на предыдущий объект. Вообще, p+=n указывает на n объектов правее, а p-=n указывает на n объектов левее в массиве объектов данного типа.
2. Если p и q – два указателя на элементы одного массива, то разность (p-q) равна количеству элементов между ними

Указатели, массивы, строки, функции

Двумерные массивы. Пусть есть описание:
int b [2] [3];
выражение b[i, j] – неправильное, b [i][j] - правильное
Двумерной массив рассматривается как массив массивов.
Размещение в памяти:



b, b[0] b[1]

int *p = b;
*(p+1) ⇔ b[0][1]
*(p+2) ⇔ b[0][2]
*(p+3) ⇔ b[1][0]

Вообще, в языке Си любое индексное выражение вида
<выражение-1> [<выражение-2>]
интерпретируется как
* (<выражение-1> + <выражение-2>)

Указатели, массивы, строки, функции

Передача массивов в качестве аргументов функций.

Одномерного:

```
f ( x )                 если хотите работать с массивом через индексы
int x[];
{ ..... }
```

или

```
f ( int *x)        если хотите работать с массивом через указатели
{ ..... }
```

Двумерного:

```
f ( x )                 если хотите работать с массивом через индексы
int x[][];
{ ..... }
```

или

```
f ( int *x[])            комбинированный способ
{ ..... }
```

Массивы указателей (пример описания):

char *p[]; p - это массив указателей, т.к. приоритет операции [] выше, чем *

char (*p)[]; а вот здесь p - это указатель на массив (имени у массива нет)

Указатели, массивы, строки, функции

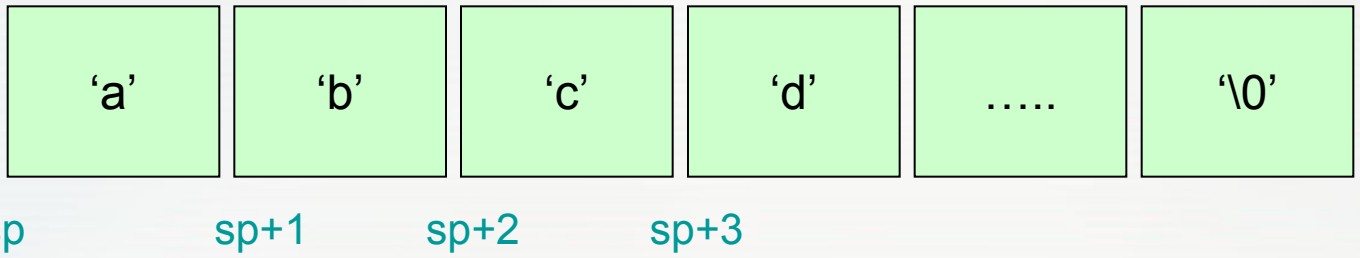
Строки и указатели

Строка есть массив символов, оканчивающийся символом '\0'.

Пусть есть такое описание:

static char s[] = "abcdef … z";

char *sp = s;



Имя строки есть синоним адреса её первого элемента.

Пример в/в строк:

используем две функции: char* gets (char*) и puts (char*)

```
{
        char buffer[81], *p;
        p = gets (buffer);
        puts (buffer);
        puts (p);
}
```

Указатели, массивы, строки, функции

Пример: вычисление длины строки

```
int strlen (char *s)          используем первый принцип адресной арифметики
{
    int n;
    for ( n = 0; *s != '\0'; s++ ) n++;    просто подсчитаем количество символов
    return n;
}
```

или можно так:

```
int strlen (char *s)          используем второй принцип адресной арифметики
{
    char *p = s;
    while ( *p )  p++;        признак конца строки '\0'
    return ( p – s );         есть арифметический ноль, он же есть «ложь»
}
```

Указатели, массивы, строки, функции

Пример: копирование строк

```
int strcpy (char *to, char *from)
{
    while ( ( *to = *from ) != '\0' )
    {  to++;  from++;  }
}
```

указатель to должен адресовать область памяти, достаточную для размещения исходной строки, которая адресуется указателем from

или можно проще:

```
int strcpy (char *to, char *from)
{
    while (  *to++ = *from++  ) ;
}
```

признак конца строки '\0' есть арифметический ноль, он же есть «ложь»

Здесь опять использован тот принцип, что в языке Си любое индексное выражение вида

<выражение-1> [<выражение-2>]

интерпретируется как

* (<выражение-1> + <выражение-2>)

Указатели, массивы, строки, функции

Аргументы командной строки

Запуская исполняемый файл в DOS, можно указать для него ряд аргументов, например:

a.exe один два три

Операционная система сформирует массив указателей на строки символов (лексемы), которые и появились в командной строке, и передаст их в функцию main.

Количество лексем – это аргумент argc, массив указателей на лексемы – аргумент argv.

В нашем случае:

argv [0] -> “a.exe”
argv [1] -> “один”
argv [2] -> “два”
argv [3] -> “три”

пример обработки аргументов командной строки:

```
main ( int argc, char *argv [] )
{
    while ( --argc )
        printf ( “ %s%c “, *++argv, (argc > 1) ? ‘ ‘ : ‘\n’ );
}
```

Указатели, массивы, строки, функции

Указатели на функции

Рассмотрим три описания:

char* f(); char (*g) (); char * (* (*h) ( )) [];

Здесь f – функция, g – указатель на функцию, h – тоже указатель на функцию, которая возвращает указатель на массив указателей на символы.

Пример:

решение нелинейного уравнения f(x) = 0 упрощенным методом Ньютона

```
#include <math.h>
double solve ( double ( *f ) (double), double guess )
{
double x;
    do { x = guess;  guess = x – ( *f ) ( x ); }
    while ( fabs ( x – guess ) > 1.0e-10 ) ;
    return guess;
}

void main ( )
{
    printf ( " %f   %f \n", solve ( sin, 0.1 ), solve ( cos, 1.0 ) ) ;
}
```

итерационная формула:

$x_{n+1} = x_n – f(x_n) / f'(x_n)$

пока $| x_{n+1} – x_n | > e$

для простоты положили $f'(x_n) = 1$

решим уравнения

$sin(x) = 0$ и $cos(x) = 0$

Резюме: Имя функции есть синоним адреса точки входа в функцию.