Указатели

Зачем нужны указатели

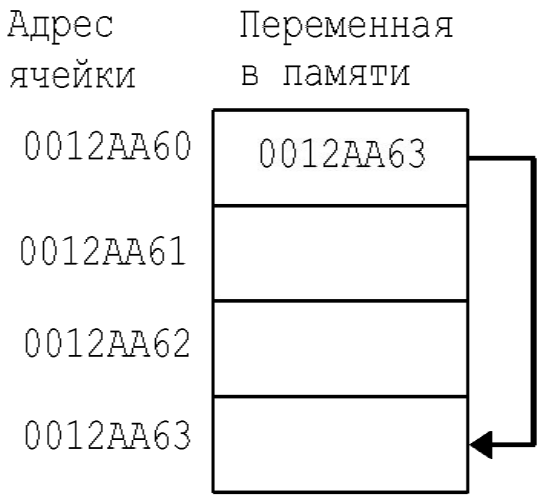
Существует три причины, по которым невозможно написать хорошую программу без использования указателей:

- указатели позволяют функциям изменять свои аргументы
- с помощью указателей осуществляется динамическое распределение памяти;
- указатели повышают эффективность многих процедур.

Кроме того, использование указателей при обращении к элементам массива или структуры делают программу более эффективной.

Что такое указатель

- *Указатель* (pointer) – переменная, в которой хранится адрес другого объекта. Обычно – это адрес другой переменной.
- Если одна переменная содержит адрес другой, то говорят, что первая переменная ссылается на вторую, как это изображено на рисунке.



Объявление указателя

Переменная, содержащая адрес ячейки памяти, должна быть объявлена как *указатель*.

Объявление указателя состоит из имени базового типа, символа * и имени переменной.

В общем виде объявление указателя записывается в виде:

тип_указателя *имя_указателя;

Базовый тип указателя определяется типом переменной, на которую он может ссылаться. Им может быть любой допустимый тип.

Например,

```
int *pointer, *mas[10];
float *pointer;
```

Операторы для работы с указателями

Оператор

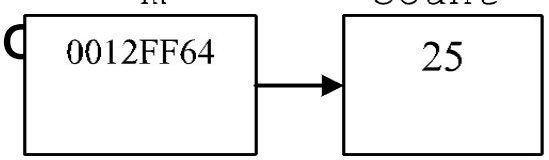
m = &count;

присваивает переменной m адрес переменной count. Этот адрес относится к ячейке памяти, которую занимает переменная count. Если переменная занимает несколько ячеек памяти, ее адресом считается адрес первого байта.

Если m содержит адрес переменной count, то оператор

q = *m;

присваивает переменной q значение переменной



Предположим, переменная count хранится в ячейке памяти под номером 0012FF64, а ее значение равно 25. Тогда переменной m будет присвоено значение 0012FF64.

Инициализировать указатель можно

- используя адрес уже объявленной переменной с помощью оператора взятия адреса &
- выделяя под него память функцией malloc()

Пример

```
#include "stdafx.h"
#include "conio.h"
void main() {
int someVariable = 4; //объявляем и инициализируем переменную someVariable
int *pointer; //объявляем указатель
pointer = &someVariable;  //инициализируем его адресом переменной someVariable
*pointer=*pointer+1; // изменяем значение, находящееся по адресу, на который ссылается
    указатель pointer
printf("Текущее значение переменной someVariable = %d\n", someVariable);
printf("Текущее значение переменной *pointer = %d", *pointer);
getch();
}
```

Выведет строки:

Текущее значение переменной someVariable = 5

Текущее значение переменной*pointer = 5

Присваивание указателей

Указатель можно использовать в правой части оператора присваивания для присваивания его значения другому указателю. Если оба указателя имеют один и тот же тип, то выполняется простое присваивание, без преобразования типа. В следующем примере

```
#include <stdio.h>
int main(void)
{
  int x = 99;   int *p1, *p2;
  p1 = &x;  p2 = p1;
printf("Значение по адресу p1 и p2: %d %d\n", *p1, *p2); /* печать значение x дважды */
printf("Значение указателей p1 и p2: %p %p", p1, p2);  /* печать адреса x дважды */
  return 0;
}
```

после присваивания

```
p1 = &x;
p2 = p1;
```

оба указателя (p1 и p2) ссылаются на x. То есть, оба указателя ссылаются на один и тот же объект. Программа выводит на экран следующее:

```
Значения по адресу p1 и p2 : 99 99
Значения указателей p1 и p2: 0063FDF0 0063FDF0
```

Преобразование типа указателя

Указатель можно преобразовать к другому типу

Эти преобразования бывают двух видов:

- с использованием указателя типа void *
- без его использования.

В языке С допускается присваивание указателя типа void * указателю любого другого типа (и наоборот) без явного преобразования типа указателя

Тип указателя void * используется, если тип объекта неизвестен. Например, использование типа void * в качестве параметра функции позволяет передавать в функцию указатель на объект любого типа, при этом сообщение об ошибке не генерируется. Также он полезен для ссылки на произвольный участок памяти, независимо от размещенных там объектов. Например, функция размещения malloc()возвращает значение типа void *, что позволяет использовать ее для размещения в памяти объектов любого типа.

В отличие от void *, преобразования всех остальных типов указателей должны быть всегда явными

Т.е должна быть указана операция приведения типов

(тип *)

В языке C++ требуется явно указывать преобразование типа указателей, в том числе указателей типа void *.

Однако следует учитывать, что преобразование одного типа указателя к другому может вызвать непредсказуемое поведение программы.

```
#include <stdio.h>
int main(void)
{
  double x = 100.1, y;
  int  *p;
  /* В следующем операторе указателю на целое p
     присваивается значение, ссылающееся на double. */
  p = (int *) &x;
  /* Следующий оператор работает не так, как ожидается. */
  y = *p; /* attempt to assign y the value x through p */
  /* Следующий оператор не выведет число  100.1. */
  printf("Значение x равно: %f (Это не так!)", y);
  return 0;
}
```

Адресная арифметика

В языке С допустимы только две арифметические операции над указателями:

- суммирование
- вычитание

Предположим, текущее значение указателя p1 типа int * равно 2000.

Предположим также, что переменная типа int занимает в памяти 2 байта.

Тогда после операции увеличения

p1++;

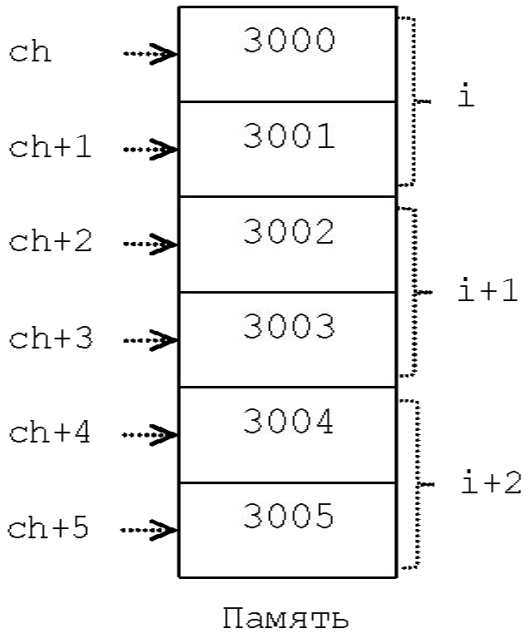
указатель p1 принимает значение 2002, а не 2001. То есть, при увеличении на 1 указатель p1 будет ссылаться на следующее целое число. Это же справедливо и для операции уменьшения. Например, если p1 равно 2000, то после выполнения оператора

p1--;

значение p1 будет равно 1998.

Операции адресной арифметики подчиняются следующим правилам. После выполнения операции увеличения над указателем, данный указатель будет ссылаться на следующий объект своего базового типа. После выполнения операции уменьшения — на предыдущий объект

```
char *ch = (char *) 3000;
int *i = (int *) 3000;
```



Применительно к указателям на char, операций адресной арифметики выполняются как обычные арифметические операции, потому что длина объекта char всегда равна 1. Для всех указателей адрес увеличивается или уменьшается на величину, равную размеру объекта того типа, на который они указывают. Поэтому указатель всегда ссылается на объект с типом, тождественным базовому типу указателя. На рисунке приведен пример размещения в памяти переменных char (слева) и int (справа) (предполагается, что длина целочисленной переменной равна 2 байтам).

К указателям можно добавлять целые числа или вычитать из них целые числа. Выполнение оператора

p1 = p1 + 12;

"передвигает" указатель p1 на 12 объектов в сторону увеличения адресов.

Кроме суммирования и вычитания указателя и целого, разрешена еще только одна операция адресной арифметики: можно вычитать два указателя. Благодаря этому можно определить количество объектов, расположенных между адресами, на которые указывают данные два указателя; правда, при этом считается, что тип объектов совпадает с базовым типом указателей. Все остальные арифметические операции запрещены.

Сравнение указателей

Стандартом С допускается сравнение двух указателей. Например, если объявлены два указателя p и q, то следующий оператор является правильным:

```
if(p < q) printf("p ссылается на меньший адрес, чем q\n");
```

Однако, сравнение указателей может оказаться полезным, только тогда, когда два указателя ссылаются на общий объект, например, на массив.

Указатели и массивы

Доступ к элементам массива с помощью адресной арифметики

Рассмотрим следующий фрагмент программы:

```
char str[80], *p1;
p1 = str;
```

Имя массива без индекса возвращает адрес первого элемента массива. Поэтому здесь p1 указывает на первый элемент массива str. Обратиться к пятому элементу массива str можно с помощью любого из двух выражений:

```
str[4]
```

или

```
* (p1+4)
```

Обращение к элементам массива

В языке С существуют два метода обращения к элементу массива:

- адресная арифметика
- индексация массива

Стандартная запись массивов с индексами наглядна и удобна в использовании, однако с помощью адресной арифметики иногда удается сократить время доступа к элементам массива. Поэтому адресная арифметика часто используется в программах, где существенную роль играет быстродействие.

```
/* Индексация указателя s как массива. */
void putstr(char *s) {
  register int t;
  for(t=0; s[t]; ++t) putchar(s[t]);
}
/* Использование адресной арифметики. */
void putstr(char *s) {
  while(*s) putchar(*s++);
}
```

Пример работы с массивами через указатели

```
#include "stdafx.h"
#include <stdlib.h>
#include <conio.h>
#define SIZE 10
 int main(void) {
int mass[SIZE], *p, *first,*max; first=mass;
for (p=mass; p<mass+SIZE; p++) scanf("%d", p);
p=mass; max=first;
while (p<first+SIZE) {
        if (*p>*max) max=p;
        p++;
}
printf("max = %d", *max);
 for (p=first; p<first+SIZE; p++)
printf("%d ", *p);
 getch();
  return 0;
}
```

Индексация указателей на многомерные массивы

Например, если a – это указатель на двухмерный массив целых размерностью 10×10, то следующие два выражения эквивалентны:

a == &a[0][0]

Более того, к элементу (0,4) можно обратиться двумя способами:

либо указав индексы массива: a[0][4],

либо с помощью указателя: *((int*)a+4).

Аналогично для элемента (1,2): a[1][2] или *((int*)a+12).

В общем виде для двухмерного массива справедлива следующая формула:

a[j][k] эквивалентно *((базовый_тип*)a+(j*кол-во_столбцов)+k)

Правила адресной арифметики требуют явного

- преобразования указателя на массив в указатель на

Массивы указателей

Как и обычные переменные, указатели могут быть собраны в массив. В следующем операторе объявлен массив из 10 указателей на объекты типа int:

```
int *x[10];
```

Для присвоения, например, адреса переменной var третьему элементу массива указателей, необходимо написать:

```
x[2] = &var;
```

В результате этой операции, следующее выражение принимает то же значение, что и var:

```
*x[2]
```

Для передачи массива указателей в функцию используется тот же метод, что и для любого другого массива: имя массива без индекса записывается как формальный параметр функции.

Например, следующая функция может принять массив x в качестве аргумента:

```
void display_array(int *q[])
{
  int t;
  for(t=0; t<10; t++)
    printf("%d ", *q[t]);
}
```

Необходимо помнить, что q – это не указатель на целые, а указатель на массив указателей на целые.

Указатели и строки

Указатели и строки

Большинство операций языка Си, имеющих дело со строками, работают с указателями. Рассмотрим, например, приведенную ниже бесполезную, но поучительную программу:

```
/* Указатели и строки */
#define PX(X) printf("X = %s; значение = %u; &X = %u\n",X,X,&X)
main( )
{
    static char *mesg = "Сообщение";
    static char *copy;
    copy = mesg;
    printf("%s\n",copy);
    PX(mesg);
    PX(copy);
}
```

Массив и указатель: различия

Возможны два способа объявления массива:

static char heart[ ]="Я люблю язык Си!";

char *head = "Я люблю язык Pascal!";

Основное отличие состоит в том, что указатель heart является константой, в то время как указатель head - переменной. Посмотрим, что на самом деле дает эта разница.

Во-первых, и в том и в другом случае можно использовать операцию сложения с указателем:

```
for(i=0;i<7;i++)
        putchar(* (heart+i));
putchar('\n');
for(i=0;i<7;i++)
        putchar(* (head+i));
putchar('\n');
```

В результате получаем:

Я люблю

Я люблю

Но операцию увеличения можно использовать только с указателем:

```
while ((*head) != '\0')     /* останов в конце строки */
putchar(*(head++)); /* печать символа и перемещение указателя */
```

В результате получаем:

Я люблю язык Pascal!

Предположим, мы хотим изменить `head` на `heart`. Можно так:

```
head=heart; /* теперь head указывает на
массив heart */
```

но теперь можно и так

```
heart = head; /* запрещенная конструкция */
```

Ситуация аналогична `x = 5` или `5 = x`. Левая часть оператора присваивания должна быть именем переменной. В данном случае `head = heart`, не уничтожит строку про язык Си, а только изменит адрес, записанный в `head`.

Вот каким путем можно изменить обращение к head и проникнуть в сам массив:

```
heart[13] = 'C';
```

или

```
*(heart+8)='C';
```

Обратите внимание, **переменными** являются **элементы массива,** но **не имя**!

Массивы указателей часто используются при работе со строками

Например, можно написать функцию, выводящую нужную строку с сообщением об ошибке по индексу num:

```
void syntax_error(int num)
{
  static char *err[] = {
    "Нельзя открыть файл\n",
    "Ошибка при чтении\n",
    "Ошибка при записи\n",
    "Некачественный носитель\n"
  };
  printf("%s", err[num]);
}
```

- Массив err содержит указатели на строки с сообщениями об ошибках. Здесь строковые константы в выражении инициализации создают указатели на строки. Аргументом функции printf() служит один из указателей массива err, который в соответствии с индексом num указывает на нужную строку с сообщением об ошибке. Например, если в функцию syntax_error() передается num со значением 2, то выводится сообщение Ошибка при записи.