Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №12

Указатели

Идея использования адресов в программах очень стара. Самые первые программисты имели хорошее представление об адресном пространстве программы. Написанные ими программы на машинном языке или языке ассемблера выполняли непосредственные и эффективные пересылки данных, расположенных по одному адресу, в другую адресуемую область.

При появлении языков более высокого уровня программисту был предоставлен высший уровень абстракции. Удобные средства, такие как имена переменных, индексирование массивов, функции и подпрограммы, освободили программиста от необходимости думать на низком уровне машинных адресов.

Язык Си предлагает лучшее из обоих подходов. Язык Си дает возможность манипулировать адресами посредством переменных типа указатель. Эффективность и гибкость, предоставляемая указателями, в сочетании с высокой мобильностью языка Си - вот две причины частого использования языка Си вместо машинного языка или языка ассемблера там, где наиболее важна скорость выполнения программы.

Указатель — это переменная, содержащая адрес.

Указатель должен быть объявлен также, как объявляются обычные переменные, но перед идентификатором должен стоять символ *****.

Пример 1

```
int          *p_i;          /* указатель на целое типа int*/
float        *p_f;          /*указатель на вещественное типа float */
struct  str  *p_s;          /*указатель на структуру  str*/
```

Для корректного использования адресной арифметики важно объявлять указатель на тот тип данных, с которым он будет использоваться в программе.

Для чего используются указатели?

- Для эффективного доступа к данным.
- Для разработки гибких программ.
- Для изменения переменных, передаваемых в функцию.
- Для работы с динамически распределяемой памятью.
- Для доступа по аппаратно-фиксированным адресам в системных программах.

Использование указателей

Указатель должен быть объявлен соответствующим образом:
тип *идентификатор;

Пример 1

```
int num1 = 3, num2 = 6, *p;
```

В прим. 1 идентификатор **p** объявляется как указатель на целое. Это означает, что, как только он будет проинициализирован, его значением будет адрес целого. Целые и указатели на целое могут быть объявлены в одном операторе. Все указатели должны быть проинициализированы.

Операция косвенного доступа

***операнд**

Операция косвенного доступа (*****) является унарной операцией, результат которой — значение объекта, на который указывает операнд. Тип результата определяется типом указателя. Приоритет этой операции определяется по таблице приоритетов операций языка Си . В качестве операнда должно использоваться именующее выражение.

Пример 2

На рис. 1 приведена программа, для которой показано содержание переменных, располагаемых на стеке.

Пусть адреса переменных **num1**, **num2** и указателя p соответственно равны 500, 504 и 510.

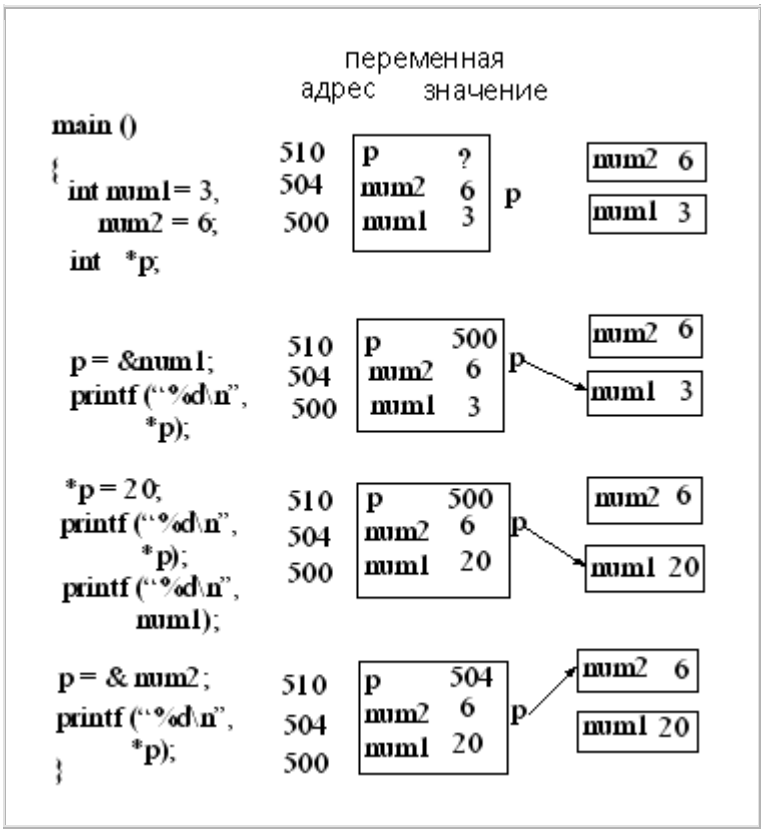

Рис. 1. Использование указателей

Вывод программы:

```
3
20
20
6
```

В прим. 2 сначала указателю **p** присваивается адрес переменной **num1**, после чего он "указывает на" **num1**. После этого доступ к значению переменной **num1** может быть выполнен либо непосредственно через идентификатор (**num1**), либо косвенно через указатель(***p**).

Адресная арифметика

Адресная арифметика языка Си включает в себя следующие арифметические операции над указателями.

- Указателю можно присвоить значение указателя того же типа или адрес объекта того же типа.
- Целое число можно сложить с указателем или вычесть из него.
- Указатель можно сравнивать с другим указателем.
- Указатель можно вычесть из другого указателя.
- Указателю можно присвоить значение нулевого указателя — константы **NULL**.
- Указатель можно сравнивать с нулевым указателем.

Другие арифметические операции над указателями недопустимы.

Константа **NULL** объявляется в файле **<stdio.h>**. Нулевой указатель часто используется для организации сложных конфигураций данных, например, связанных списков и деревьев. Другой пример использования нулевого указателя: результат выполнения возвращающих указатель функций, сообщающий о неудаче.

Чтение или запись по адресу 0 в языке Си не разрешены и могут быть причиной аварийного завершения программы.

При использовании адресной арифметики важно использовать указатели с правильным типом. Если указатель будет использован для хранения адресов целых типа **int**, он должен быть объявлен как указатель на **int**. Если указатель будет содержать адреса объектов типа **double**, он должен быть объявлен как указатель на **double**, и т.д.

В языке Си последовательный доступ к данным через указатели осуществляется очень просто путем использования возможностей, предоставляемых адресной арифметикой. Когда целое число прибавляется к указателю (или вычитается из него), оно предварительно умножается на размер объекта (в байтах), на который ссылается указатель.

Пример 1

```
int a, *p_int;
...
char ch, *p_char;
p_int = &a;
p_char = &ch;
p_int++;
p_char++;
```

Пусть переменная **a** имеет адрес 500, а переменная **ch** — 520. Тогда после инициализации значением указателя **p_int** будет 500, а **p_char** соответственно 520. Операция инкрементации применительно к указателям увеличит **p_int** на 4, а **p_char** на 1.

Адресную арифметику особенно удобно использовать при обработке массивов.

Типичные приемы работы с указателями

В примерах приводятся три (из множества) возможных варианта функции **strlen()**, которая возвращает количество символов в строке вплоть до признака конца строки **'\0'**, но не включая его.

Пример 2

```
int strlen (register const char *s)
{
  register char *p;
  p = s;
  while (*p != '\0')
       p++;
  return p-s;
}
```

Этот вариант функции наиболее нагляден.

Пример 3

```
int strlen (register const char *s)
{
 register char *p;
 p = s;
 while(*p)
      p++;
 return p-s;
}
```

Данный вариант равносилен предыдущему, т.к. выражение ***p** будет истинно, пока его значение не нулевой символ (не символ **'\0'**).

Пример 4

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

В данном варианте используется выражение, применяемое особенно часто: **`*p++`**. Так как унарные операции право-ассоциативные (связываются с операндами справа налево), это выражение интерпретируется как **`*(p++)`**. Это значит, что будет увеличен указатель **`p`**, но не значение выражения **`*p`**. Поскольку операция инкрементации **`++`** употребляется в постфиксной форме (после операнда), указатель **`p`** не увеличивается, пока его значение не использовано. Общее действие состоит в использовании ссылки **`*p`** и последующем увеличении указателя **`p`**. В операторе возврата из **`(p-s)`** вычитается 1. Это делается потому, что после инкрементации **`p`** указывает на следующий за нулевым символом байт.

Указатель на void

Тип "*указатель на неопределенный тип*" (указатель на void) — обобщенный указательный тип. Значение указателя любого типа может быть преобразовано и присвоено указателю на тип void и наоборот без изменения его значения. К такому указателю нельзя применять операцию косвенного доступа. Для применения этой операции указатель на **`void`** надо привести к определенному типу.

Пример 1

```
int main()
{
  int a_function(void *);
  char name[50] = "William";
  int num = 95;

  a_function(name);                  /* передан указатель на char */
  a_function(&num);                  /* передан указатель на int */
}

int a_function(void *vptr)
{
  ...
}
```

В общем случае, отличном от рассмотренного примера, результат преобразования значений и присваивания указателей различных типов непредсказуем.

Массивы указателей

Массив указателей - это массив адресов.
Массив указателей объявляется также как и прочие массивы:

тип *идентификатор[целочисленное выражение]

Пример 1

```
char buffer[30001];
/* Текстовый буфер, заканчивающийся '\0'*/
char *line_num[2001];
/*Максимальное число строк - 2000*/



void number_lines(void)
 {
 int i;
 char *p;
 line_num[0]=buffer;
 for(p=buffer, i=1; *p != '\0'; p++)
   if(*p == '\n')
     if(*(p+1) != '\0')
       line_num[i++] = p+1;
 line_num[i] = NULL;
 }
```

Предположим, что массив **buffer** уже заполнен текстом. Функция **number_lines()** занесет начальные адреса каждой из находящихся в буфере строк в массив символьных указателей **line_num**, то есть **line_num[0]** будет указывать на первую строку, **line_num[1]** будет указывать на вторую строку и т.д. Список указателей на строки завершается нулем.

Инициализация массива указателей при объявлении

Пример 2

```
/*пример сканера для идентификации команд */
#include <stdio.h>
#include <string.h>
int is_keyword(char *);
char *keyword[] = {
                    "append",
                    "find",
                    "list",
                    "remove",
```

```
                    "replace",
                    "substitute",
                    (char *)NULL
                      };
/*возвращает индекс команды, иначе -1*/
int is_keyword(char *)
  {
  int i;
  for (i=0; keyword[i] != NULL; i++)
    if(strcmp(str, keyword[i]) == 0)
      return i;
  return -1;
    }
```

Строковые константы располагаются в области данных. Их начальные адреса будут использоваться при инициализации массива указателей. Массив указателей использует меньше памяти, чем требовалось бы для двумерного массива символов. При использовании двумерного массива символов пришлось бы найти самое длинное ключевое слово и указать его длину, увеличенную на 1 за счет символа - признака конца строки, в качестве второй размерности массива. Например, имеется 30 ключевых слов и самое длинное содержит 7 символов. Тогда двумерный массив должен быть объявлен следующим образом:

```
char keyword[30][8];
```

В итоге появилось бы много неиспользуемых байтов, так как многие ключевые слова содержат менее 7 символов.

Двукратные указатели

Двукратный указатель - это переменная, которая содержит адрес указателя, то есть это указатель на указатель.

Двукратные указатели часто используются в системных программах. Трехкратные указатели встречаются редко. Язык Си допускает так же использование многократных указателей (например, `****p`). Однако лучше избегать этого.

Объявление двукратного указателя:

```
тип **идентификатор;
```

Пример 1

```
/*пример сканера для идентификации команд */
#include <stdio.h>
#include <string.h>
int is_keyword(char *);
char *keyword[] = {
                    "append",
                    "find",
                    "list",
                    "remove",
```

```
                    "replace",
                    "substitute",
                    (char *)NULL
                    };
/*возвращает индекс команды, иначе -1*/
int is_keyword(char *)
 {
  char **p;
  for (p=keyword; *p != NULL; p++)
    if(strcmp(str, *p) == 0)
      return p-keyword;
  return -1;
 }
```

Массив **keyword** - это массив указателей на символ, который инициализируется адресами строковых констант. Такие указатели обычно не изменяются во время выполнения программы. Двукратный указатель **p** используется для перебора элементов массива.

p содержит адрес одного из элементов массива **keyword**, ***p** - это адрес строки, записанный в этом элементе массива, а ****p** (в примере не используется) указывает на первый символ в этой строке.

Функции, возвращающие указатель

Функция может возвращать указатель. При этом она должна быть определена и объявлена соответствующим образом. Например, если функция возвращает указатель на целое, ее определение обычно имеет вид:

```
int *f(void)
{
. . .
}
```

Кроме того, функция должна быть объявлена в некоторой вызывающей функции или выше всех таких функций по тексту программы.

Пример 1

```
main ()
{
int *f(void);
...
}
```

В прим. 1 объявление известно внутри функции main().

Пример 2

```
int *f(void);
main ()
{
...
}
```

В прим. 2, подобно внешней переменной, объявление известно всем функциям, расположенным ниже его в исходном тексте программы. Объявление может предваряться ключевым словом **extern**.

Пример 3

```
#include "local.h"
main ()
{
...
}
```

Прим. 6 напоминает, что объявления функций часто находятся в файлах заголовков, область действия этих объявлений такая же, как в прим. 2.

Пример 4

```
/* main () - передает адрес массива, содержащего город и улицу,
в функцию getstreet(), которая возвращает указатель на улицу*/
#include <stdio.h>
int main ()
{
  char *getstreet (char *);
  static char info[] = "Москва, Арбат";

  printf ("Улица %s\n", getstreet(info));
}
/* Предполагается, что p указывает на строку вида "Город, Улица" */
char *getstreet(char *p)
{
 while ( *p != ',')
    p++;
 return p + 2;
}
```

Вывод:

```
Улица Арбат
```

Указатели на функцию

Указатель может быть использован для ссылки на функцию, точно также как и для ссылки на переменную.

Пример 1

```
int main ()
{
int (*funptr)(int, int);      /*объявляется указатель на функцию*/
int maxfun(int, int);         /*объявляется функция,  на которую будет ссылаться указатель*/
...
funptr = maxfun;              /*инициализация указателя*/
c=(*funptr)(a,b);             /* c=maxfun(a,b);*/
...
}
int maxfun(int i, int j)
{
return (i>j)?i:j;
}
```

Пример 2

Пусть функции, выполняющие различные запросы пользователя, определены в другом месте.

```
int main()
{
int append(void), find(void), list(void);
int remove(void), replace(void), substitute(void);

static int (*command[])(void) = {
                                  append,
                                  find,
                                  list,
                                  remove,
                                  replace,
                                  substitute
                                 };
int i, (*fp)(void);
char string[80];
...
scanf("%s", string);
...
```

```
i = is_keyword(string);
if (i==-1)
   exit(0);
fp=comand[i];                /*инициализирует указатель на функцию*/
(*fp)();                     /*вызов функции (*command[i])()*/
...
}
```

Чтобы использовать имя функции в выражении, функция должна быть явно объявлена. В функции main() объявляются имена функций. Затем инициализируется массив указателей **command[]** адресами функций. Объявляется указатель на функцию **fp**. Читается запрос пользователя и передается функции **is_keyword()**, возвращающей индекс соответствующего элемента массива адресов функций. Этот индекс используется для инициализации указателя **fp**, определяющего требуемую функцию.

Доступ к аргументам командной строки

В операционной среде, обеспечивающей поддержку языка Си, имеется возможность передать аргументы или параметры запускаемой программе с помощью командной строки.

Пример 1

Это командная строка (символ $ не входит в командную строку, это "приглашение" операционной системы).

```
$ a.out file1 file2
```

Аргументы командной строки — это множество слов, из которых состоит командная строка.

В функцию main() при передаче управления всегда передаются два аргумента. В программе они могут игнорироваться и, следовательно, не обрабатываться. Чтобы получить доступ к аргументам командной строки, в определении функции **main()** в круглых скобках после идентификатора**main** должны быть объявлены два формальных параметра.

argc (сокращение от argument count) - первый параметр функции **main()**, имеет смысл счетчика аргументов. Это количество слов, на которые разбивается символами промежутков командная строка.

argv (сокращение от argument vector- вектор аргументов) — второй параметр функции **main()**. Во время выполнения программы на языке Си слова из командой строки размещены в виде символьных строк в программном адресном пространстве. В этой же памяти запоминается и завершаемый нулем (**NULL**) массив адресов (указателей) этих строк.

Пример 2

```
#include <stdio.h>
int main (int argc, char *argv[])
{
  int i;
  printf ("number of words: %d\n", argc);
  for (i=0; i<argc; i++)
     printf ("%s\n", argv[i]);
```

}

В прим. 2 **argv** объявляется как массив указателей на символьные значения. **argv[0]** – это адрес, по которому хранится первое слово (в прим. 1 это **a.out**); **argv[1]** — адрес второго слова и т.д.

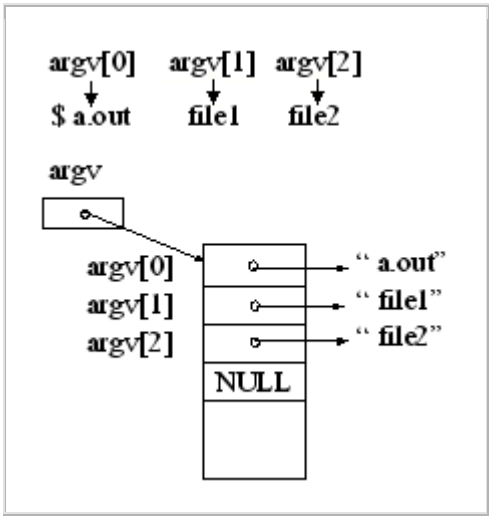

Рис. 1. Пример командной строки

Использование двукратного указателя для доступа к командной строке

Второй параметр функции main() argv — это двукратный указатель.

Пример 3

```
/* печатает аргументы командной строки, используя argv как двукратный указатель*/
#include <stdio.h>
int main(int argc, char **argv)
 {
 for(; *argv != NULL; argv++)
  printf("%s\n", *argv);
 }
```