Лабораторная работа №9

Указатели. Указатели на массивы.

Цель работы: Изучение теоретических сведений и получение практических навыков по работе с указателями.

Теоретические сведения

Каждая переменная в языке Си – это объект, имеющий имя и адрес в области памяти, по которому хранится значение этой переменной. Таким образом, чтобы получить значение некоторой переменной необходимо либо обратиться к значению переменной через имя этой переменной, либо получить доступ к участку памяти, который выделен для ее хранения. На рисунке 1 представлены два уровня представления любой переменной.

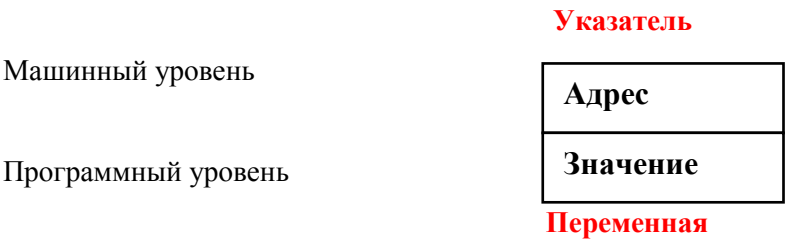

Рисунок 1 – Уровни представления переменной: машинный и программный.

Таким образом, каждая переменная располагается в памяти компьютера и каждая ячейка памяти имеет адрес, доступ к которому можно получить через оператор **&**. Рассмотрим следующий пример программу, позволяющая выводить на экран адрес, по которому хранится переменная:

```
#include <stdio.h>
int main ()
{
        int  var1;
        char var2[10];
        printf("Address of var1 variable: %x\n", &var1);
        printf("Address of var2 variable: %x\n", &var2);
        return 0;
}
```

Указатель – это переменная, значение которой является адресом другой переменной. Так же как и значение обычной переменной значение указателя необходимо определить до того, как указатель будет использован.

Синтаксис определения указателя следующий:

```
type *ptr;
```

где type – это один из базовых типов языка Си, ptr – имя указателя, а * используется для определения указателя. Таким образом, в качестве примера можно привести следующие объявления указателей:

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Единственная разница между приведенными выше указателями – тип данных, т.е. тип данных переменной, на которую он будет указывать . Значением же указателя во всех приведенных случаях будет являться шестнадцатеричная константа, представляющая собой адрес ячейки памяти (см. Рисунок 2).

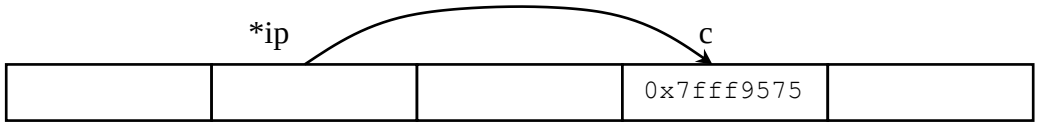

Рисунок 2 – Связь между указателями и переменными.

Работа с указателями

Существуют несколько операций, которые можно выполнять с помощью указателей. Для того чтобы использовать указатель необходимо последовательно выполнить следующие действия:

а) объявить указатель;

б) установить указатель на адрес переменной;

в) получить доступ к адресу, на который указывает указатель.

Таким образом, унарный оператор * возвращает значение переменной, размещенной по адресу. Рассмотрим иллюстрацию работы с указателями на примере программы:

```
#include <stdio.h>
int main ()
{
      int  var = 20;   /* явное объявление переменной */
```

```
    int  *ip;        /* объявление указателя */
    ip = &var;  /* помещаем адрес переменной в указатель*/
    printf("Address of var variable: %x\n", &var);
    /* Адрес, который хранится в переменной */
    printf("Address stored in ip variable: %x\n", ip );
    /* Доступ к значению переменной var */
    printf("Value of *ip variable: %d\n", *ip );
    return 0;
}
```

Фактически унарный указатель * - это ***оператор косвенного доступа***. Применяя его к указателю он выдает объект, на который настроен данный указатель. Предположим, что x и y имеют тип int, а ptr является указателем на int. Покажем, каким образом объявляются указатели и как используются операторы & и *.

```
int x = 1, y = 2, z[10];
int *ptr; /* ptr - указатель на int */
ip = &x; /* теперь ptr указывает на x */
y = *ptr; /* y теперь равен 1 */
*ptr = 0; /* x теперь равен 0 */
ptr = &z[0]; /* ptr теперь указывает на z[0] */
```

Если ptr указывает на x целочисленного типа, то *ptr можно использовать в любом месте, где допустимо применение x, например,

```
*ptr = *ptr + 10;
```

увеличивает *ptr на 10.

Унарные операторы * и & имеют более высокий приоритет, чем арифметические операторы, так что присваивание

```
y = *ptr + 1;
```

берет то, на что указывает ptr, и добавляет к нему 1, а результат присваивает переменной y. Аналогично

```
*ptr += 1;
```

увеличивает на единицу то, на что указывает ptr; те же действия выполняют

```
++*ptr;
```

и

```
(*ptr)++;
```

В последней записи скобки необходимы, поскольку если их не будет, увеличится значение самого указателя, а не то, на что он указывает. Это обусловлено тем, что унарные операторы * и ++ имеют одинаковый приоритет и порядок выполнения – справа налево.

Указатель NULL

Хорошим тоном при программировании является использование значения NULL для указателей, которые пока не ассоциированы с определенным адресом переменной. Указатели ассоциированные со значением NULL называется нуль-указателями. Рассмотрим пример работы с такими указателями.

```
#include <stdio.h>
int main ()
{
      int  *ptr = NULL;
      printf("The value of ptr is : %x\n", ptr  );
      return 0;
}
```

Указатели и аргументы функций

Поскольку в Си функции в качестве своих аргументов получают значения параметров, нет прямой возможности, находясь в вызванной функции, изменить переменную вызывающей функции. Пусть нам нужна функция `swap`, меняющая местами значения двух переменных. Однако недостаточно написать `swap(a, b)`, где функция `swap` определена следующим образом:

```
void swap(int x, int y) /* НЕВЕРНО */
{
      int temp;
      temp = x;
      x = y;
      y = temp;
}
```

Поскольку `swap` получает лишь копии переменных `a` и `b`, она не может повлиять на переменные `a` и `b` той функции, которая к ней обратилась. Чтобы получить желаемый эффект, вызывающей функции надо передать указатели на те значения, которые должны быть изменены:

```
swap(&a, &b);
```

Так как оператор & получает адрес переменной, &a есть указатель на a. В самой же функции swap параметры должны быть объявлены как указатели, при этом доступ к значениям параметров будет осуществляться косвенно.

```
void swap(int *px, int *py) /* перестановка *px и *py */
{
      int temp;
      temp = *px;
      *px = *py;
      *py = temp;
}
```

Аргументы-указатели позволяют функции осуществлять доступ к объектам вызвавшей ее функции и дают возможность изменить эти объекты.

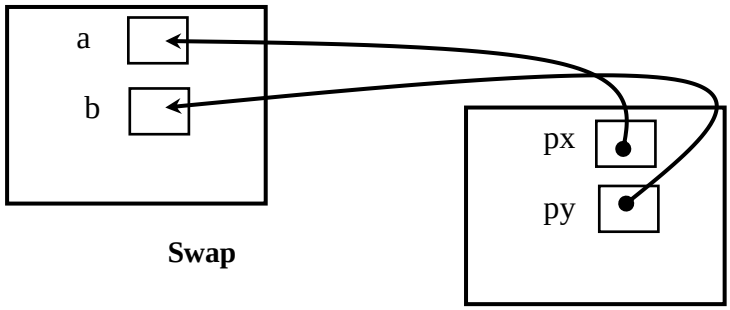

Рисунок 3 – Иллюстрация работы функции swap.

Указатели и массивы

В Си существует явная связь между указателями и массивами. Эта связь настолько тесная, что указатели и массивы лучше рассматривать совместно. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть также выполнен с помощью указателей. В общем случае указатели будут работать быстрее. Объявление массива a содержащего 10 элементов будет выглядеть следующим образом:

```
int a[10];
```

и представлять собой блок из 10 последовательных объектов с именами a[0], a[1], ..., a[9]. Использование в программе идентификатора a[i] отсылает нас к i-му элементу массива.

Рассмотрим как с помощью указателя pa типа int, объявленного

```
int *pa;
```

получить доступ к элементам массива a. В результате следующего присваивания:

```
pa = &a[0];
```

указатель pa будет настроен на нулевой элемент a, иначе говоря, указатель pa будет содержать адрес элемента a[0].

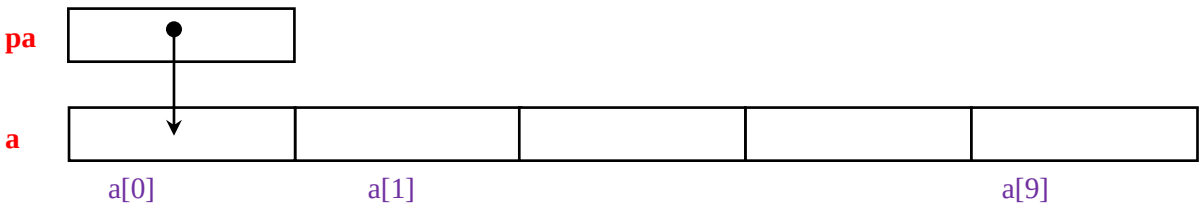

Рисунок 4 –Указатель и массив.

Между индексированием и арифметикой с указателями существует очень тесная связь. По определению значение переменной или выражения типа массив есть адрес нулевого элемента массива. После присваивания

```
pa = &a[0];
```

pa и a имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание pa = &a[0] можно также записать в следующем виде:

```
pa = a;
```

Теперь рассмотрим как с помощью указателя pa получить доступ к остальным элементам массива. Указатель pa настроен на нулевой элемент массива, добавление к единицы pa+1, по определению, указывает на следующий элемент. Таким образом, pa+i указывает на i-й элемент после pa, а указатель pa-i – на i-й элемент перед pa. Таким образом, если pa указывает на a[0], то указатель *(pa+1) содержит адрес a[1] элемента, a+i – адрес a[i], в то время как *(pa+i) содержит адрес a[i]. Все замечания верны безотносительно к типу и размеру массива a. Смысл слов «добавить 1 к указателю», как и смысл любой арифметики с указателями, состоит в том, чтобы pa+1 указывал на следующий объект (иными словами, на 1-й после pa).

Кроме того, a[i] можно записать как *(a+i). Вычисляя a[i], компилятор сразу преобразует его в *(a+i); указанные две формы записи эквивалентны. Из этого следует, что полученные в результате применения оператора & записи &a[i] и a+i также будут эквивалентными. С другой стороны, если pa – указатель, то его можно использовать с индексом, т.е. запись pa[i] эквивалентна записи *(pa+i). Короче говоря, элемент массива можно изображать как в виде указателя со смещением, так и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель – это переменная, поэтому можно написать pa = a

или pa++. Но имя массива не является переменной, и записи вроде a=pa или a++ не допускаются.

Следующая конструкция с присваиванием

```
x = *pa;
```

копирует содержимое a[0] в переменную x.

Указатели в списке формальных параметров функции

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес.

Мы можем воспользоваться отмеченным фактом и написать функцию strlen, вычисляющую длину строки.

```
/* strlen: возвращает длину строки */
int strlen(char *s)
{
      int n;
      /* цикл от первого элемента строки до последнего (последний
      * элемент строки – нулевой символ '\0')
      * с шагом в один символ
      */
      for (n = 0; *s != '\0' ; s++)
            n++;
      return n;
}
```

Так как переменная s – указатель, к ней применима операция ++, s++ не оказывает никакого влияния на строку символов функции, которая обратилась к strlen. Просто увеличивается на 1 некоторая копия указателя, находящаяся в личном пользовании функции strlen. Пример вызова:

```
int n;
n = strlen("Здравствуй, мир");
```

или

```
char str[] = "Здравствуй, мир"
int n;
n = strlen(str);
```

Формальные параметры `char s[]` и `char *s` в определении функции эквивалентны. Мы отдаем предпочтение последнему варианту, поскольку он более явно сообщает, что `s` есть указатель. Если функции в качестве аргумента передается имя массива, то она может рассматривать его так, как ей удобно – либо, как имя массива, либо как указатель, и поступать с ним соответственно. Она может даже использовать оба вида записи, если это покажется уместным и понятным.

Функции можно передать часть массива, для этого аргумент должен указывать на начало подмассива. Например, если `a` – массив, то в записях `f(&a[2])` или `f(a+2)` функции `f` передается адрес подмассива, начинающегося с элемента `a[2]`. Внутри функции `f` описание параметров может выглядеть как

```
f(int arr[]) {…}
```

или

```
f(int *arr) {…}
```

Следовательно, для `f` тот факт, что параметр указывает на часть массива, а не на весь массив, не имеет значения. Поэтому, если есть уверенность, что элементы массива существуют, то возможно индексирование и в «обратную» сторону по отношению к нулевому элементу; выражения `arr[-1]`, `arr[-2]` и т. д. не противоречат синтаксису языка и обращаются к элементам, стоящим непосредственно перед `arr[0]`. Разумеется, нельзя «выходить» за границы массива и тем самым обращаться к несуществующим объектам.

Символьные указатели (строки)

Строковая константа, написанная в виде: `"Я строка"` есть массив символов. Во внутреннем представлении этот массив заканчивается нулевым символом `'\0'`, по которому программа может найти конец строки. Число занятых ячеек памяти на одну больше, чем количество символов, помещенных между двойными кавычками.

Чаще всего строковые константы используются в качестве аргументов функций, как, например, `printf`:

```
printf("здравствуй, мир\n");
```

Когда такая символьная строка появляется в программе, доступ к ней осуществляется через символьный указатель; `printf` получает указатель на начало массива символов. Точнее, доступ к строковой константе осуществляется через указатель на ее начальный элемент.

Строковые константы нужны не только в качестве аргументов функций. Если, например, переменную pmessage объявить как

```
char *pmessage;
```

то присваивание

```
pmessage = "now is the time";
```

поместит в нее указатель на символьный массив, при этом сама строка не копируется, копируется лишь указатель на нее. Операции для работы со строкой как с единым целым в Си не предусмотрены.

Существует важное различие между следующими определениями:

```
char amessage[] = "now is the time"; /* массив */
char *pmessage = "now is the time"; /* указатель */
```

`amessage` – это массив, имеющий такой объем, что в нем как раз помещается указанная последовательность символов и `'\0'`. Отдельные символы внутри массива могут изменяться, но `amessage` всегда указывает на одно и то же место памяти. В противоположность ему `pmessage` есть указатель, инициализированный так, чтобы указывать на строковую константу. А значение указателя можно изменить, и тогда последний будет указывать на что-либо другое. Кроме того, результат будет неопределен, если вы попытаетесь изменить содержимое константы.

Контрольные вопросы

1. Что такое указатель?
2. Как Вы понимаете, что такое адрес?
3. Опишите схематично устройство памяти компьютера и принципы хранения переменных в ней.
4. Сколько однобайтовых ячеек необходимо для указателя в 32-битной архитектуре? В 64-битной?
5. Какая операция позволяет получить адрес объекта?
6. Если `p` объявлено, как `int *p`, то какой смысл у выражений `p = a` и `*p = a` в исходном тексте (a объявлено как `int a`)?
7. Для чего используется указатель NULL?
8. Как изменить в вызываемой функции значения переменных в вызывающей функции? Приведите пример (пример с функцией `swap` не приводить).

9. Какова связь между указателями и массивами? Если массив объявлен, как `int a[10]`, то что такое `a`?
10. Если `p` объявлено, как `int *p`, то какой смысл у выражений `p++` и `p-3` в исходном тексте?
11. Как определить, что при посимвольном просмотре строки мы дошли до ее конца?

Задание на лабораторную работу

1. Изучить теоретические сведения.
2. Создать блок-схему для выбранного преподавателем варианта задания.
3. Написать по ней программу, протестировать и, при необходимости, отладить ее.
4. Подготовить отчет о выполнении лабораторной работы.
5. Подготовить ответы на контрольные вопросы.

Варианты заданий

Часть А (Задания общие для всех)

1. Выясните, сколько байт в памяти занимают переменные самых распространенных типов и указатели на них (`char`, `short`, `int`, `long`, `float`, `double`, `long double`).
2. Оформите задачу из предыдущей лабораторной работы в виде функции, для передачи массива в функцию используйте указатель.

Часть Б (Индивидуальные задания)

1. Написать программу, создающую массив из `10` целых чисел из отрезка `[-50;50]`, которые пользователь задает с консоли. Вывести на экран весь массив и на отдельной строке – значение минимального элемента массива.

   Для обхода массива использовать указатели (запрещено обращаться к элементам массива по индексам).
2. Написать программу, которая копировала бы строку введённую пользователем с клавиатуры в новую (максимальная длина строки - `20` символов). При этом в процессе копирования должны отбрасываться все незначащие пробелы в начале и конце строки, а также несколько подряд идущих пробелов должны заменяться на один. Вывести исходную и новую строки на экран. Для обхода строк использовать указатели (запрещено использовать библиотечную функцию `strcopy`).
3. Написать программу, которая для введённой с клавиатуры строки (максимальная длина строки – `20` символов) сообщает, какая цифра в ней встречается чаще всего, либо сообщает, что цифры в строке совсем отсутствуют. Если с одинаковой частотой в строке встретилось несколько цифр, то в качестве лидера вывести

любую из подходящих цифр. Для обхода строк использовать указатели.

4. Дан символьный массив: "`If we cannot do as we would we`", необходимо "развернуть" строку при копировании. То есть, если у нас есть "`abcdef`", то после выполнения программы мы должны получить "`fedcba`". Решить задачу с помощью указателя на массив.
5. Найти среднее арифметическое из трех чисел, используя только указатели.
6. Дан массив из `10` чисел типа `double`. Написать программу, которая удваивает значения элементов в массиве. Решить задачу через указатели.
7. Найти среднее арифметическое массива, состоящего из 10 элементов, с использованием указателя.
8. Дан массив из `10` чисел типа `double`. Вывести на экран значения нулевого, второго и четвертого элементов массива, с использованием указателя.
9. Для введённой пользователем с клавиатуры строки (максимальная длина строки – `20` символов) программа должна определить, корректно ли расставлены скобки (круглые, фигурные, квадратные) или нет. Перемешивание скобок (пример: «{[}]») считается некорректным вариантом. Для решения задачи использовать указатели. **Это задание сложнее, чем остальные.**