МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический
университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Чистопольский филиал «Восток»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

ПРОГРАММИРОВАНИЕ И ОСНОВЫ АЛГОРИТМИЗАЦИИ

Индекс по учебному плану: **Б1.О.12.02**
Направление подготовки: **09.03.01 Информатика и вычислительная техника**
Квалификация: **Бакалавр**
Профиль подготовки: **Вычислительные машины, комплексы, системы и сети**
Типы задач профессиональной деятельности: **проектный, производственно-технологический**

Рекомендовано УМК ЧФ КНИТУ-КАИ

Чистополь
2023 г.
.

Лабораторная работа 1

Функции

Вы уже знакомы с некоторыми библиотечными функциями, такими как printf(), scanf(), getchar(), putchar(), gets(), sin(), cos(), ... Теперь нужно знать, как создавать свои собственные функции.

Функция - это самостоятельная единица программы, предназначенная для решения определенной задачи. Функции в языке Си играют ту же роль, какую играют функции, подпрограммы и процедуры в других языках программирования.

Программа на Си может состоять из любого количества функций, одна из которых всегда носит имя main. Выполнение программы начинается с функции main(), которая может вызывать другие функции. Те в свою очередь тоже могут вызывать какие-то функции.

Описания функций могут размещаться в одном файле или в нескольких. Рассмотрим случай, когда все функции программы описываются в одном файле. В этом случае расположить в файле описания функций можно в любом порядке. Но проще описание каждой функции поместить перед ее использованием (вызовом) в других функциях.

Описание любой функции имеет вид:

```
Заголовок функции
{  Описания локальных переменных
   Операторы
}
```

Заголовок функции имеет формат:

тип_функции имя_функции (список_параметров)

Через параметры (аргументы) происходит передача данных между функцией и вызывающей программой. В списке параметров перечисляются через запятую описания параметров (перед каждым параметром указывается тип). Имена параметров могут быть любые, т.к. это формальные параметры. Если функция не имеет параметров, то пишутся пустые скобки.

Примеры заголовков функций:

```
main()
float max (float x, float y)
int prov (char mas[], int s)
void line (ink k, int n, char simv, int ps)
```

Тип функции, указанный перед именем функции, - это тип возвращаемого функцией значения. По умолчанию предполагается тип ***int***. Если функция никакое значение не возвращает, то указывается тип ***void.***

Допускается запись заголовка функции в иной форме, когда в списке параметров перечисляются только имена, а описания параметров выносятся за скобки. Например:

```
void line (k,n,simv,ps)
      int k,n;
      char simv;
      int ps;
```

Но такая форма записи уже устарела (и имеет недостатки).

Значение функции передается в вызывающую программу с помощью оператора возврата ***return.***

Пример описания функции:

```
/*  функция определения наибольшего из двух чисел  */
float  max ( float x, float y )
{  if (x>y) return x;
   else return y;
}
/*  2-й вариант  */
float  max ( float x, float y )
{  float z;    /*  z=max(x,y)  */
   z = (x>y) ? x : y ;
   return z;
}
```

Вы можете вызвать свою функцию в любом выражении, указав ее имя и аргументы (фактические параметры):

имя_функции (аргумент_1, аргумент_2, ...)

Аргументом может быть идентификатор, константа и выражение. Типы аргумента и соответствующего параметра в заголовке функции должны совпадать. Например, оператор

f = max(a,b) - max(c,d);

содержит два вызова приведенной выше функции max. При первом обращении функции max передаются значения переменных a и b, она возвращает наибольшее из этих чисел, которое подставляется вместо указателя функции max(a,b). При втором вызове функции max формальным параметрам x и y присваиваются соответственно значения фактических параметров c и d. Оператор return возвращает наибольшее из этих значений в точку вызова функции.

Вызов функции без параметров имеет вид: **имя_функции ().**

Использование указателей при передаче параметров

Обмен информацией между вызывающей программой и функцией осуществляется благодаря параметрам. Если данные входные и функцией не изменяются, то передавать в списке параметров нужно значения данных. Сложнее вернуть вызывающей программе выходные данные, если их несколько (если результатом выполнения функции является одно единственное значение, то оно обычно, как вы уже видели, в число параметров не включается, а является значением функции). Если функция должна вернуть вызывающей программе нес колько значений, в этом случае передавать ей в качестве фактических параметров нужно **адреса** переменных. Вспомните вызов функции scanf(). Если, например, нужно ввести значения двух переменных x и y типа int, вы должны написать:

scanf ("%d %d",&x,&y);

где &x,&y - адреса переменных x и y. Точно так же при обращении к своей функции для выходных параметров нужно указывать адреса.

В описании функции соответствующие формальные параметры должны быть описаны как **указатели** на переменные соответствующего типа.

Допустим, функция function1() должна изменить значения переменных a и b типа float. В вызывающей программе обращение к функции нужно записать так:

function1 (&a, &b);

а заголовок функции должен иметь вид:

void function1 (float *ptr1, float *ptr2)

Здесь prt1 и ptr2 - это указатели на переменные типа float (названия их могут быть изменены - это формальные параметры). Значениями указателей являются адреса. При вызове функции function1 переменной pt1 присваивается адрес a, а ptr2=&b.

Таким образом функции становятся доступными переменные вызывающей программы. Если нужно этим переменным присвоить новые значения, допустим, с помощью операторов присваивания, то это делается так:

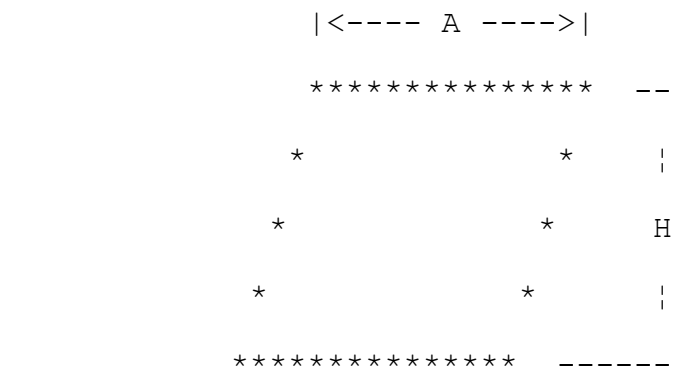
*ptr1 = ... ;

*prt2 = ... ;

Итак, если ptr1 и ptr2 являются указателями на переменные, то чтобы взять значения этих переменных, нужно перед именами указателей использовать операцию *****.

Пример

Задача. Нарисовать параллелограмм вида



при заданных значениях A, H и C, где C - смещение фигуры по горизонтали вправо относительно левой границы экрана.

В приведенной ниже программе параллелограмм вычерчивается в текстовом режиме, хотя можно (и проще) было бы сделать это в графическом режиме, используя стандартные графические процедуры Turbo C.

Программа

```
#include <stdio.h>
#include <conio.h>
/*    Функция вывода горизонтального отрезка прямой    */
void   line ( int k, int n, char simv, int ps)
        /*  Входные данные:                                          */
        /*       k    - смещение отрезка вправо,                     */
        /*       n    - длина отрезка,                               */
        /*       simv - символ заполнения отрезка,                   */
        /*       ps - признак перевода строки:                       */
        /*            если ps!=0, то нужен перевод строки после вывода*/
        /*            отрезка; если ps=0, то не нужен перевод строки  */
{    int i;    /* параметр цикла */
        /* установление левой границы отрезка */
     for ( i=0; i<k; i++ )   putchar (' ');
        /* вывод отрезка */
```

```
    for ( i=0; i<n; i++ )   putchar (simv);
    if (ps)  putchar ('\n');
}
      /*   О с н о в н а я   п р о г р а м м а          */
main()
{   int  c,    /* смещение нижнего левого угла */
         a,h,  /* основание и высота параллелограмма */
         i,    /* параметр цикла */
         cls;  /* смещение левой боковой стороны */
    printf ("Введите три числа: смещение, основание, высоту.\n");
    scanf ("%d %d %d",&c,&a,&h);
    cls=c+h-1;
    line (cls,a,'*',1);
    for (i=0; i<h-2; i++)
      { line (--cls,1,'*',0);
        line (a-2,1,'*',1);
      }
    line (c,a,'*',1);
    getch();
}
```

Порядок выполнения работы

1. Получить задание у преподавателя.

2. Составить схему и программу на Си и подобрать тесты для проверки программы на компьютере.

3. Выполнить тестирование и отладку программы на компьютере, показать результаты преподавателю.

Задания

Нарисовать заданную фигуру при заданных значениях A (основание), H (высота) и C (смещение фигуры по горизонтали вправо), используя процедуру вывода горизонтального отрезка прямой, приведенную в

примере.

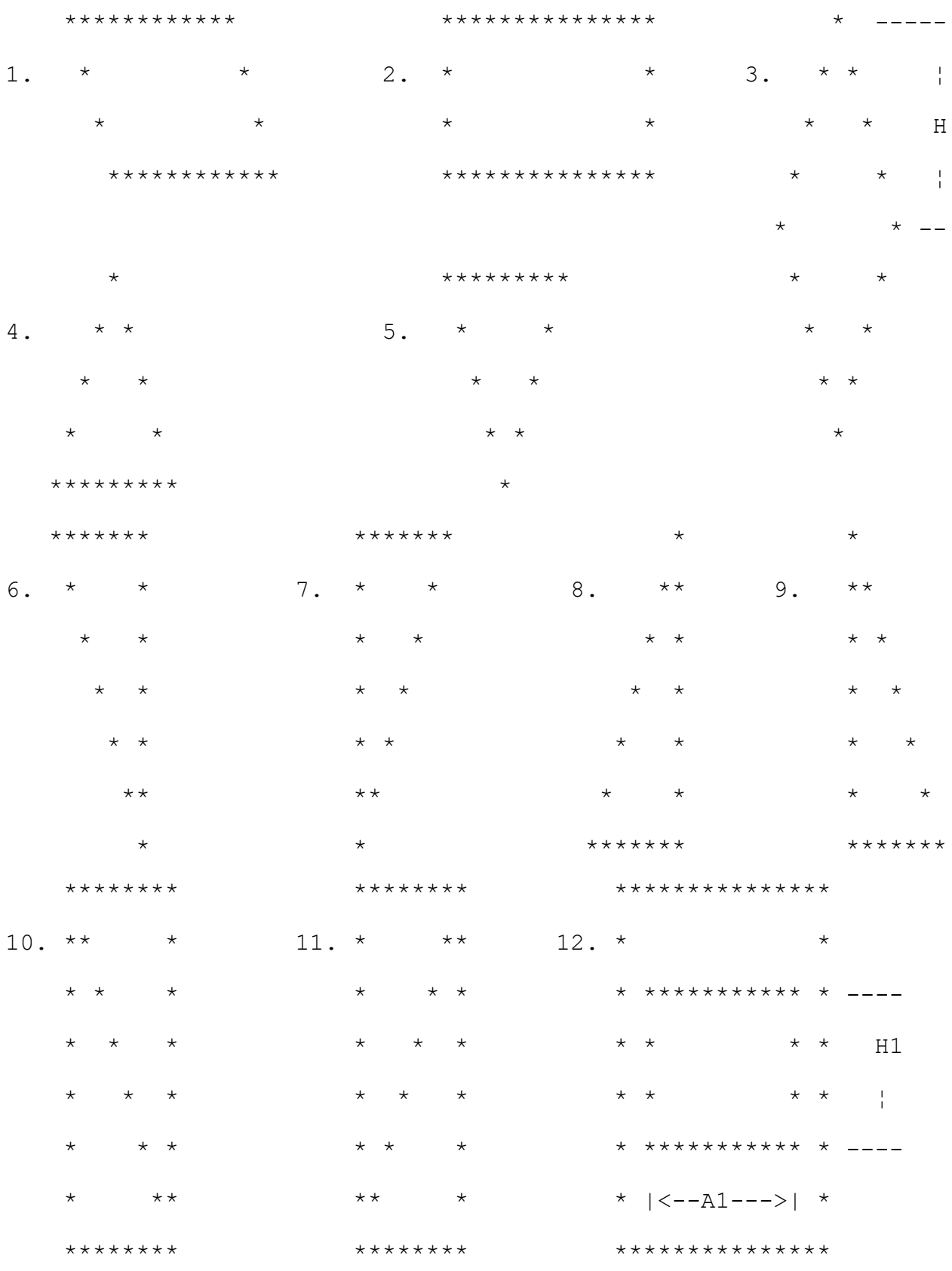
Составить несколько вариантов основной программы:

а) начертить только контур фигуры;

б) заполнить внутреннюю часть фигуры тем же символом, которым изображается контур фигуры;

в) заполнить внутреннюю часть фигуры точками.

Варианты заданий:



Примечание. Все значения A,H,C задают число выводимых символов или число позиций. В заданиях 4-5 A=2H-1, в заданиях 6-11 A=H.

Лабораторная работа 2
Символьные строки и функции обработки строк

Строка символов - это последовательность символов произвольной длины, завершающаяся нуль-символом '\0'(все биты в байте нулевые).

Строковые константы записываются в кавычках, например:

"Как Вас зовут?"

Если в программе встречается строковая константа, компилятор выделяет для нее память объемом, равным длине строки (количеству символов) + 1 (для нуль-символа). Признак конца строки '\0' добавляется автоматически.

Строковые переменные описываются либо как массивы символов, либо как указатели на символы, например:

```
char stroka[81];      /* строка длиной до 80 символов */
char *str;            /* указатель на строку */
```

В первом случае память для строки выделяется (имя массива является адресом его первого символа), во втором случае - нет (выделяется только для указателя).

При выполнении оператора

```
str= "Группа 21101";
```

указателю str присваивается адрес памяти, где размещена строковая константа "Группа 21101". Для массива такой оператор будет неверным, т.к. адрес массива - величина постоянная.

Еще раз обратите внимание, что в Си имя массива отождествляется с адресом его первого элемента. Для описанного выше массива stroka имя массива stroka и &stroka[0] идентичны. Другими словами, имя массива является указателем на первый символ, причем его нельзя изменять.

Если строка будет вводиться с клавиатуры, то лучше описать переменную как массив символов, иначе предварительно придется выделять память для строки, например, с помощью функции ***malloc().***

Функции gets() и puts()

Для ввода и вывода строк символов служат функции gets() и puts().

Функция gets() вводит с клавиатуры строку, заменяя символ "перевод строки" на нуль-символ, и помещает ее по указанному адресу. Например:

```
gets (stroka);  /* ввод строки в массив строка */
```

Функция puts() выводит указанную строку на экран. Например, оператор

```
puts(str);
```

выведет на экран строку, на которую указывает переменная str, курсор после вывода переместится на новую строку.

Функции обработки строк

Для работы со строками символов в библиотеке Turbo C имеется ряд функций, например, функция определения длины строки strlen(), копирования строк strcpy(), сцепления строк strcat(), сравнения строк strcmp(), нахождения в строке указанного символа strchr(). Прототипы таких функций находятся в файле ***string.h***, их перечень приведен в следующем разделе.

Примеры обращения к функциям обработки строк:

```
char s1[81], s2[81], s3[81];
char *s;
gets(s1);
printf ("Длина строки = %d\n", strlen(s1));
strcpy(s2,s1);    /* копирование строки s1 в массив s2  */
if ((s=strchr(s2,'a'))!=NULL)    /* есть буква 'a' в строке s2 */
     *s = 'b';     /* замена в строке s2 первой буквы 'a' на 'b' */
gets(s3);
if (strcmp(s1,s3)==0) printf ("Строки одинаковые\n");
```

Рассмотрим одну из библиотечных функций - функцию сцепления двух заданных строк strcat(). Определение функции:

```
char *strcat (char *s1, char *s2);
```

Функция копирует строку s2 (на которую ссылается указатель s2) в конец строки s1 и возвращает значение s1 - ссылку на сцепленную строку.

Работу функции можно описать так:

```
char *strcat (char *s1, char *s2)
   {     char *rs;       /* ссылка на результирующую строку   */
  rs=s1;  /* запоминание адреса начала строки s1 */
  while (*s1!='\0')   s1++;  /* поиск конца строки s1 */
       /* копирование строки s2 в конец s1 */
  while (*s2!='\0')
    { *s1=*s2; s1++; s2++; }
  *s1='\0';
   return rs;
   }
```

Как видите, функция не проверяет, достаточно ли памяти для результирующей строки. Вызывающая программа должна позаботиться об этом.

А теперь посмотрите на более компактную (но менее понятную) запись этой функции:

```
char *strcat (char *s1, char *s2)
   {     char *rs;
  rs=s1;  /* запоминание адреса начала строки s1 */
  while (*s1!='\0') s1++;          /* поиск конца строки s1 */
  while ((*s1++ = *s2++) !='\0');  /* копирование s2 в конец s1,
                                      включая нуль-символ */
  return rs;
   }
```

Даже еще можно сократить текст функции, записав второй оператор **while** короче:

```
      while (*s1++ = *s2++);
```

Каждый раз очередной символ из второй строки копируется в первую,

затем значения указателей s1 и s2 увеличиваются на 1, т.е. происходит продвижение указателей к следующим символам строк. Этот процесс повторяется до тех пор , пока не скопируется нуль-символ (т.к. выход из цикла происходит при нулевом значении выражения в скобках).

Теперь вам предлагается самим **написать одну из функций обработки строк** из списка библиотечных или дополнительных функций, указанную преподавателем. Для проверки работы вашей функции напишите драйвер (программу отладки) в виде функции main().

Пример драйвера для функции сцепления строк strcat():

```
#include <stdio.h>
/*     Тестирование ф-ции strcat    */
/*        Программа-драйвер          */
main()
  { char str1[81],str2[81];
    puts ("Введите две строки");
    gets (str1);
    gets (str2);
    if (strlen(str1)+strlen(str2) < 81)
      { puts ("Результат:");
        puts (strcat(str1,str2));
        printf ("Строки после вызова функции сцепления:\n%s\n%s\n",
                  str1,str2);
      }
    else puts ("Не хватает памяти для результирующей строки");
    getch();
  }
```

Библиотечные функции обработки строк

1. strcmp - сравнить две строки.

Определение: int strcmp (char *s1, char *s2);

0, если строки одинаковые;

Значение функции = разность кодов двух первых несовпадающих символов, если строки разные.

2. strncmp - сравнить первые n символов двух строк.

Определение: int strncmp (char *s1, char *s2, int n);

0, если первые n символов строк совпадают;

Значение функции

разность кодов символов, если первые n символов строк не совпадают.

3. strcpy - копировать строку s2 в s1.

Определение: char *strcpy (char *s1, char *s2);

Значением функции является s1 - ссылка на первую строку.

4. strncpy - копировать не более n символов строки s2 в s1.

Определение: char *strncpy (char *s1, char *s2, int n);

Значением функции является s1 - ссылка на первую строку.

5. strlen - определить длину строки (число символов без завершающего нуль-символа).

Определение: int strlen(char *s);

6. strchr - найти в строке s первое вхождение символа c.

Определение: char *strchr (char *s, char c);

Значение функции - ссылка на первый символ c в строке s или NULL (пустая ссылка), если символа нет в строке.

7. strrchr - найти в строке s последнее вхождение символа c.

Определение: char *strrchr (char *s, char c);

Значение функции - ссылка на последний символ c в строке s или NULL (пустая ссылка), если символа нет в строке.

8. strpbrk - найти в строке s1 любой из множества символов, входящих в строку s2.

Определение: char *strpbrk (char *s1, char *s2);

Значение функции - ссылка на любой символ в строке s1, имеющийся в s2, или NULL (пустая ссылка), если символов из s2 нет в s1.

9. strstr - найти в строке s1 первое вхождение строки s2.

Определение: char *strstr (char *s1, char *s2);

Значение функции - ссылка на первое вхождение s2 в s1 или NULL (пустая ссылка), если подстроки s2 нет в s1.

10. strncat - сцепить две строки s1 и s2, причем из второй строки копировать не более n символов.

Определение: char *strncat (char *s1, char *s2, int n);

Значением функции является s1 - ссылка на результирующую строку (n символов строки s2 копируется в конец строки s1).

Дополнительные функции обработки строк

11. strrsub - найти в строке s1 последнее вхождение строки s2.

Определение: char *strrsub (char *s1,char *s2);

Значение функции - ссылка на последнее вхождение s2 в s1 или NULL (пустая ссылка), если подстроки s2 нет в s1.

12. delchr - удалить в строке s первое вхождение символа c.

Определение: char *delchr (char *s,char c);

Значением функции является s - ссылка на строку.

13. delrchr - удалить в строке s последнее вхождение символа c.

Определение: char *delrchr (char *s,char c);

Значением функции является s - ссылка на строку.

14. delnchr - удалить в строке s n первых символов.

Определение: char *delnchr (char *s,int n);

Значением функции является s - ссылка на строку.

15. delchrn - удалить в строке s все символы, кроме n первых .

Определение: char *delchrn (char *s,int n);

Значением функции является s - ссылка на строку.

16. delstr - удалить в строке s1 первое вхождение строки s2.

Определение: char *delstr (char *s1,char *s2);

Значением функции является s1 - ссылка на первую строку.

17. chngchar - заменить в строке s каждый символ c1 на символ c2.

Определение: char *chngchar (char *s,char c1,char c2);

Значением функции является s - ссылка на строку.

18. chngstr - заменить в строке s первое вхождение строки s1 на строку s2.

Определение: char *chngstr (char *s,char *s1,char *s2);

Значением функции является s - ссылка на строку.

Порядок выполнения работы.

1. Получите задание у преподавателя.

2. Составьте функцию и драйвер, подберите тесты для проверки функции на ЭВМ.

3. Если аналогичная функция есть в библиотеке Turbo C, выполните тестирование библиотечной функции с помощью вашего драйвера, добавив директиву #include <string.h> (разумеется, свою функцию пока включать в программу не нужно). Затем выполните тестирование и отладку своей функции (результаты должны быть такими же, как и при выполнении библиотечной функции).

Лабораторная работа 3

Рекурсивная функция

Цели:

1. Изучить особенности работы рекурсивных функций на языке программирования Си++.

2. Написать программу, решающую указанную задачу при помощи рекурсии.

Варианты задания:

1. Подсчитать количество цифр в заданном натуральном числе, используя рекурсивную подпрограмму.

2. Описать функцию C(m,n), где 0≤ m≤ n, для вычисления биномиального коэффициента C_n^m по следующей формуле:

$C_n^0 = C_n^n = 1; C_n^m = C_{n-1}^m + C_{m-1}^{n-1}$ при 0< m< n

используя рекурсивную подпрограмму

3. Описать рекурсивную функцию Root(f, b, ε), которая методом деления отрезка пополам находит с точностью ε корень уравнения f(x) = 0 на отрезке [a, b] (считать, что ε > 0, a < b, f(a} • f(b) < 0 и f(x)- непрерывная и монотонная на отрезке [a, b]).

4. Описать функцию min(x) для определения минимального элемента линейного массива x, введя вспомогательную рекурсивную функцию min1(k), находящую минимум среди последних элементов массива x, начиная с k-го.

5. Описать рекурсивную логическую функцию Simm(S, i, j), проверяющую, является ли симметричной часть строки S, начинающаяся i-м и кончающаяся j-м ее элементами.

6. Составить программу вычисления наибольшего общего делителя двух натуральных чисел, используя рекурсивную подпрограмму.

7. Составить программу нахождения числа, которое образуется из данного натурального числа при записи его цифр в обратном порядке, используя рекурсивную подпрограмму. Например, для числа 1234 получаем ответ 4321.

8. Составить программу перевода данного натурального числа в p-ичную систему счисления ($2 \le p \le 9$) ,используя рекурсивную подпрограмму.

9. Дана символьная строка, представляющая собой запись натурального числа в p-ичной системе счисления ($2 \le p \le 9$). Составить программу перевода этого числа в десятичную систему счисления , используя рекурсивную подпрограмму.

10. Составить программу вычисления суммы:

$1! + 2! + 3! + \ldots + n!$ ($n \le 20$).

Примечание. Тип результата значения функции - LongInt.

11. Составить программу вычисления суммы:

$2! + 4! + \ldots + n!$ ($n \le 20$, n - четное).

Примечание: Тип результата значения функции — LongInt.

12. Дано n различных натуральных чисел. Напечатать все перестановки этих чисел.

Лабораторная работа 4

Структуры

Если массив – это совокупность однотипных элементов, то структура объединяет логически связанные данные разных типов.

При использовании структурных переменных вначале нужно описать тип (или шаблон) структуры, а затем структурные переменные. Тип структуры задает порядок следования отдельных элементов (полей) в структуре и их типы. Описание структурного типа имеет вид:

```
struct  имя_типа
{  Описание элементов
};
```

Точка с запятой в конце описания обязательна. Например:

```
struct  BOOK
{  char author[40];       /* автор книги */
   char  name {50};       /* название */
   int year;              /* год издания */
   int pages;             /* число страниц */
};
```

Здесь *BOOK* - это имя типа структуры, а не переменная. При описании типа память под структуру не выделяется.

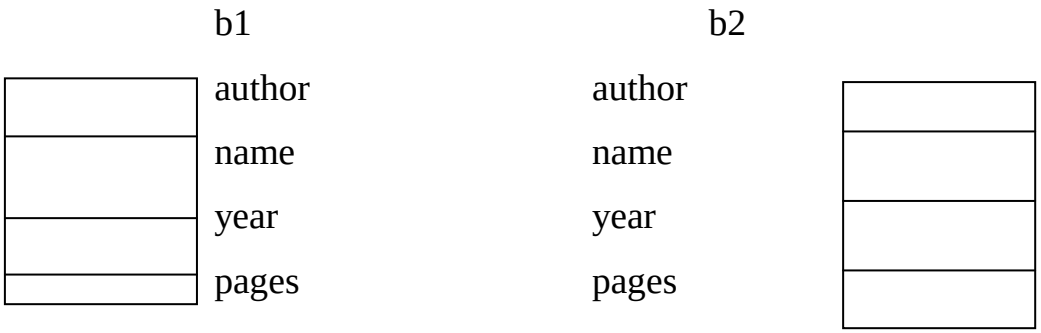
Описание структурной переменной имеет вид:

```
struct   имя_типа    имя_переменной;
```

Например:

```
struct  BOOK     b1,b2;
struct  BOOK     mb[100];   /* массив из 100 структур типа  BOOK  */
struct  BOOK      *p2 = &b2;   /* указатель на структуру типа  BOOK,
                                  ссылающийся на переменную b2      */
```

Каждая структурная переменная будет состоять из полей, перечисленных при описании типа структуры.



При объявлении структурных переменных можно их инициализировать, например:

struct BOOK b3 = { "Толстой Л.Н.", "Война и мир", 1995, 950};

Для однотипных структур определена операция присваивания.

Пример:

b1=b3;

mb[0]=b3;

Другие операции целиком над структурами выполнять нельзя. Например, сравнение структур или ввод/вывод нужно выполнять поэлементно.

Для обращения к отдельным элементам (полям) структурных переменных используется операция выбора "." или "->" :

имя_переменной . имя_элемента

имя_указателя -> имя_элемента

Пример 1

```
/* ввод значения структурной переменной  b2 типа BOOK*/
gets(b2.author);
gets (b2.name);
scanf("%d%d", &b2.year, &b2.pages);
```

Пример 2

```
struct BOOK  *p3 = &b3;
printf ("%s, %s, %d г., %d с.\n",  p3->author, p3->name,
p3->year, p3->pages);
```

Результат на экране:

Толстой Л.Н., Война и мир, 1995 г., 650 с.

Если структурные переменные используются только в одной функции программы, то можно совместить описание переменных с описанием типа. При этом имя типа можно не задавать, например:

```
struct
{ char author[40];        /* автор книги */
  char  name {50};        /* название */
  int year;               /* год издания */
  int pages;              /* число страниц */
 }  b1, b2, *p1;
```

Память под структурную переменную может быть выделена динамически, при выполнении программы с помощью функции *malloc()*, например:

```
struct   BOOK  *p4;
p4 = malloc (sizeof (struct  BOOK));
```

Такая структура имени не имеет; обращаются к ней с помощью указателя p4, например:

```
gets (p4->name);
```

Работа с файлами

Файл – это поименованная область на диске, содержащая какую-либо информацию, например, текст программы, данные для программы, документ.

Файлы бывают текстовые и двоичные (бинарные). Текстовые файлы – это файлы, которые создаются или которые можно просмотреть с помощью текстовых редакторов. В операционной системе MS DOS текстовые файлы представляют собой последовательность символьных строк. Каждый символ занимает один байт. Строка заканчивается двумя символами: «возврат каретки» (с кодом 13) и «перевод строки» (с кодом 10).

Двоичные файлы содержат информацию во внутреннем представлении. Примером двоичного файла является exe-файл, содержащий программу в машинных командах. Прикладная программа тоже может создать двоичный

файл, записав в него данные в том виде, в каком они хранятся в памяти (к примеру, типа int).

При работе с файлами программа на языке Си должна вначале открыть каждый файл с помощью функции ***fopen()***. При этом для каждого файла создается структура типа ***FILE,*** содержащая всю необходимую информацию о файле. Тип такой структуры с именем FILE определен в файле stdio.h. В программе необходимо для каждого файла описать указатель на структуру типа FILE и присвоить ему значение, возвращаемое функцией fopen().

Для чтения информации из файла служат функции:

fscanf() – форматированный ввод,

fgets() – чтение одной строки,

fgetc() – чтение одного символа,

fread() – ввод заданного числа байтов (символов).

Для записи информации в файл используются функции:

fprintf() – форматированный вывод,

fputs() – вывод строки,

fputc() – вывод одного символа,

fwrite() – вывод заданного числа байтов (символов).

Обычно к файлу применяется последовательный доступ: чтение из файла и запись в файл происходят последовательно, например, в цикле по одной строке либо блоками байтов. При этом указатель текущей позиции файла каждый раз автоматически сдвигается на начало следующей строки или следующего блока. Однако возможен и прямой (произвольный) доступ к файлу. Он используется, когда файл состоит из записей одинаковой (фиксированной) длины. Для прямого доступа к файлу служит функция ***fseek()***. Она устанавливает указатель текущей позиции файла в указанное место.

После завершения работы с файлом его следует закрыть с помощью функции ***fclose()***.

Некоторые функции доступа к файлам

fopen – открытие файла.

Прототип функции:

FILE * fopen (char * fname, char * mode);

Первый параметр fname задает имя открываемого файла, второй – режим открытия файла или вид его обработки. Параметр mode может задаваться в виде:

“r” – чтение файла,

“w” – запись в файл (если файл существует, он стирается),

“a” – добавление информации в конец файла,

“r+” – чтение и запись.

Если файл открывается для добавления информации в конец файла, то указатель текущей позиции файла устанавливается в конец файла. В остальных случаях указатель устанавливается на начало файла.

Дополнительно во втором параметре можно указать символ ***t*** (текстовый режим) или ***b*** (двоичный режим). Например, “rb+” означает, что файл открывается для чтения и записи в двоичном режиме. Режимы отличаются только обработкой символов перехода на новую строку. В текстовом режиме при чтении строки комбинация символов «возврат каретки» и «перевод строки» заменяется одним символом перевода строки (при записи в файл выполняется обратное преобразование). В двоичном режиме эти преобразования не выполняются. По умолчанию файл открывается в текстовом режиме.

Функция fopen() создает структуру типа FILE с информацией о файле и возвращает адрес этой структуры. При попытке открыть несуществующий файл для чтения или чтения и записи функция возвращает пустую ссылку NULL. Если открывается несуществующий файл для записи или добавления в конец файла, файл создается.

При открытии файла с ним связывается область памяти, называемая буфером ввода/вывода. Чтение информации из файла осуществляется блоками, равными размеру буфера. Функции чтения читают данные из буфера. При выводе информации в файл данные вначале помещаются в буфер и только после заполнения буфера записываются в файл. Буферизация повышает скорость обмена информацией между оперативной памятью и внешними устройствами.

При аварийном завершении программы выходной буфер может быть не выгружен и возможна потеря данных.

Пример

```
FILE *fout, *fmod;     /* указатели на выходной и модифицируемый файлы */
char fname[13];        /* имя модифиц. файла */
fout = fopen (“f1.txt”, “w”);
puts(“Введите имя модифицируемого файла”);
gets (fname);
if  ((fmod = fopen(fname, “r+”) ==NULL)
{  puts (“Файл в текущем каталоге не найден”);
   exit (1);
}
```

fclose – закрытие файла.

Функция имеет один параметр – указатель на файл.

Пример

```
fclose (fout);
fclose (fmod);
```

При закрытии файла информация из выходного буфера выгружается в файл. Поэтому следует все файлы закрывать явно.

fgets – чтение строки файла.

Прототип функции:

char * fgets (char *s, int n, FILE *f);

Функция считывает символы из файла, на который ссылается указатель f, в строку с указателем s. Параметр n задает предельную длину считываемой строки: читается не более (n-1) символов. В конец строки добавляется нулевой байт (‘\0’). В отличие от функции gets() символ перевода строки сохраняется (если он был прочитан).

Функция возвращает адрес строки s или значение NULL при достижении конца файла.

Пример

```
char str[81];
FILE *f;
  …
while (fgets(str,81,f))
{   /* обработка строки */
   .
   .
   .
}
```

Пример решения задачи

Задача. Входной файл **st.txt** содержит сведения о сдаче студентами группы экзаменационной сессии. Каждая запись файла содержит фамилию и инициалы студента (15 символов) и пять оценок (5 символов) и завершается символом ”перевод строки”. Напечатать список студентов с указанием среднего балла каждого студента.

Программа:

```
   /*----------------------------------------------------*/
   /*  Печать среднего балла каждого студента */
   /*----------------------------------------------------*/
#include <stdio.h>
struct STUDENT
{ char fio[15];   /* фамилия и.о. */
```

```
  char oc[7];     /* 5 оценок + '\n' + '\0' */
};
void main()
{ FILE *f;           /* указатель на входной файл */
  struct  STUDENT  tz;  /* текущая запись файла */
  int i,
      s;             /* сумма оценок */
  if ((f= fopen("st.txt","r")) == NULL)
  { puts ("Файл st.txt не найден");
    return;
  }
  puts ("\nФамилия и.о.   Ср.балл");
  puts ("-----------------------------");
  while (fgets(&tz,sizeof(struct STUDENT),f)!=NULL)
  { for (i=0,s=0; i<5; i++)
      s+=tz.oc[i]-'0';
    tz.fio[14]='\0';
    printf("%s  %.1f\n", tz.fio, (float)s/5);
  }
  fclose(f);
  getch();
}
```

Пример входного файла st.txt:

Анисимов А.И. 54435

Берхеев П.В. 55445

Вавилова С.Н. 45343

Задание

Введите и выполните программу. Создайте файл **st.txt** для проверки программы и снова запустите программу.

Измените файл **st.txt** так, чтобы поле фамилии занимало 20 позиций, а оценок было 4. Соответственно измените программу (для удобства задайте именованные константы). Проверьте результат ее работы. Объясните программу.

Измените программу, чтобы выводился список только тех студентов, у которых

а) средний балл ниже 4;

б) средний балл выше 4 и нет двоек;

в) только хорошие и отличные оценки;

г) есть двойки;

д) все пятерки;

е) больше одной двойки.

Добавьте определение среднего балла группы.

Лабораторная работа 5

Стеки

Стек - это динамическая структура данных, из которой элементы удаляются в порядке, обратном их поступлению ("последним пришел - первым вышел").

Хранить стек можно в виде вектора и указателя на последний записанный в стек элемент (указателя верхушки стека). Если стек пустой, указатель равен -1. При добавлении нового элемента в стек указатель увеличивается на 1, а при удалении элемента из стека -уменьшается на 1.

Пример

Задача. Дано арифметическое выражение длиной до 80 символов, оканчивающееся пробелом. Выражение содержит целые числа без знака и знаки операций +, -, *, /. Вычислить значение выражения.

Например, входной текст: 130+25*3-160/20*6 .

Результат: 157.

Метод решения задачи основан на использовании стека операндов, операций и приоритетов. Операции * и / имеют одинаковый приоритет, причем более высокий, чем операции + и - . При просмотре выражения происходит выделение очередного операнда (числа), преобразование его из символьной формы в целочисленную и запись в стек полученного числа, следующей за ним операции и присвоенного ей приоритета. Пока в стеке более одного элемента и приоритет последней операции оказывается не выше приоритета предыдущей операции в стеке, происходит выполнение соответствующих операций над двумя последними операндами стека и удаление ненужных элементов из стека. В конце концов, когда просмотр выражения закончится, первый элемент стека и будет искомым результатом.

В программе стек реализован в виде трех параллельных массивов.

Программа:

```
#include <stdio.h>
#include <conio.h>
```

```
   /*-------------------------------------------------------*/
   /*   Функция выделения числа и преобразования его из     */
   /*    символьной формы представления в целочисленную  */
   /*-------------------------------------------------------*/
int chislo ( char text [], int *i )
     /* Входные данные:                                     */
     /*      text - символьная строка,                      */
     /*      *i - индекс первой цифры числа в строке text.  */
     /* Выходные данные:                                    */
     /*      *i - индекс следующего после числа символа.    */
     /* Функция возвращает целое число                      */
{   int c=0;   /* возвращаемое значение  */
    while (text[*i]>='0' && text[*i]<='9')
    { c=c*10+(text[*i]-'0');
      (*i)++;
    }
    return c;
 }
      /*-----------------------------------*/
      /*       Главная функция             */
      /*-----------------------------------*/
int  main()
{   char  text [80]; /* вх. текст - арифм. выражение */
    int   i ;            /* индекс массива text */
    float  opd [3];  /* стек операндов   */
    char   opc [3];  /* стек операций    */
    int    pr [3] ;     /* стек приоритетов */
    int    j = -1 ;     /* указатель стека */
    printf ("\nВведите арифметическое выражение.\n");
    gets (text);
```

```
    i=-1;
    do
    {  i++;
        /* запись числа, операции и ее приоритета в стек */
       opd[++j]=chislo(text,&i); opc[j]=text[i];
       switch  (text[i])
       {
          case '+':
          case '-': pr[j]=1; break;
          case '*':
          case '/': pr[j]=2; break;
          case ' ' /* пробел */: pr[j]=0;
       }
       while  (j>0 && pr[j]<=pr[j-1])
       {      /* выполнение соответствующей операции */
          switch  (opc[j-1])
          {  case '+' : opd[j-1]=opd[j-1]+opd[j]; break;
             case '-' : opd[j-1]=opd[j-1]-opd[j]; break;
             case '*' : opd[j-1]=opd[j-1]*opd[j]; break;
             case '/' : opd[j-1]=opd[j-1]/opd[j];
          }
             /* удаление выполненной операции из стека */
          opc[j-1]=opc[j];  pr[j-1]=pr[j];  j=j-1;
       }
    }
    while (text[i]!=' ');
    printf ("Результат:%.2f\n",opd[0]);
    getch();
}
```

Результаты тестирования

Введите арифметическое выражение.

9/4-12

Результат:-9.75

Введите арифметическое выражение.

10*2/5+6/3

Результат:6.00

Введите арифметическое выражение.

130+25*3-160/20*6

Результат:157.00

Введите арифметическое выражение.

2345

Результат:2345.00

Задания

1. Дано арифметическое выражение длиной до 20 символов, оканчивающееся пробелом. Выражение содержит однобуквенные идентификаторы и знаки операций +, -, *, /. Преобразовать выражение в обратную польскую запись, используя стек операций и приоритетов.

Пример.

Вх. текст: a*b+c/d-e

Результат: ab*cd/+e-

2. Дано арифметическое выражение в постфиксной (обратной польской) записи длиной до 20 символов, оканчивающееся пробелом. Получить эквивалентную последовательность элементарных присваиваний, содержащих одну арифметическую операцию (используя стек операндов).

а) В левой части операторов присваивания использовать вспомогательные переменные R1,R2,R3,.

Пример.

Вх. текст: abc*+def+/-

Результат: R1:=b*c

R2:=a+R1

R3:=e+f

R4:=d/R3

R5:=R2-R4

б) В левой части операторов присваивания использовать вспомогательные переменные R1,R2,R3,..., сократив до минимума число этих переменных.

Пример.

Вх. текст: abc*+def+/-

Результат: R1:=b*c

R1:=a+R1

R2:=e+f

R2:=d/R2

R2:=R1-R2

3. Дано арифметическое выражение длиной до 30 символов, оканчивающееся знаком равенства. Выражение содержит знаки операций +, -, *, / и однозначные целые числа и представлено в обратной польской записи. Вычислить значение выражения, используя стек операндов.

Пример.

Вх. текст: 345+2*63/-+=19

результат

4. Дано арифметическое выражение длиной до 40 символов, оканчивающееся знаком равенства. Выражение содержит знаки операций +, -, / и однозначные целые числа. Вычислить значение выражения, используя стек операндов, операций и приоритетов.

Пример.

Вх. текст: 5-2+8/4-6/2=2

Результат

5. Дан ор.граф без циклов в виде количества вершин n<=10 и матрицы смежности.

а) Проверить, существует ли путь от вершины A к вершине B.

б) Найти какой-нибудь путь от начальной вершины к конечной, если такой существует.

Указание. Для хранения очередного пути при обходе графа использовать стек.

6. Дан орграф в виде количества вершин n<=10 и матрицы смежности.

а) Проверить, существует ли цикл, проходящий через заданную вершину А.

б) Найти какой-нибудь цикл, проходящий через начальную вершину, если такой существует.

Указание. Для хранения очередного пути при обходе графа использовать стек.

7. Дано скобочное выражение длиной до 50 символов, оканчивающееся пробелом. Напечатать попарно порядковые номера соответствующих открывающих и закрывающих скобок в выражении.

Пример.

1 4 8 10 14 19 27

Вх. текст: (a+(b+c)/(a-b)*(x+(y+2)**3))/2

Результат: 4 8

10 14

19 23

16 27

1 28

Указание. Для запоминания номеров открывающих скобок использовать стек.

8. Дана последовательность чисел, оканчивающаяся нулем.

а) Напечатать только отрицательные числа из этой последовательности, причем, если подряд идет несколько отрицательных чисел, печатать их в обратном порядке.

Пример.

Вх. последовательность: 5 -1 -20 -8 10 14 -3 5 -2 -13 -7 0

Результат: -8 -20 -1 -3 -7 -13 -2

Указание. Последовательность подряд расположенных отрицательных чисел помещать в стек.

б) Напечатать только положительные числа из этой последовательности, причем, если подряд идет несколько положительных чисел, печатать их в обратном порядке.

Пример.

Вх. последовательность: -2 5 10 20 15 -4 -6 2 -10 3 1 4 -5 0

Результат: 15 20 10 5 2 4 1 3

Указание. Последовательность подряд расположенных положительных чисел помещать в стек.

9. Дана последовательность из n слов (n<=20) в алфавитном порядке. Длина каждого слова не более 10 букв. Напечатать слова, начинающиеся с одной и той же буквы в обратном порядке, используя стек.

Пример.

Вх. последовательность:	Результат:
Август	апрель
Апрель	август
Май	море
Март	март
Море	май
Лето	

Литература

1. Немцова, Т. И. Программирование на языке высокого уровня. Программирование на языке С++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2023. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное образование). - ISBN 978-5-8199-0699-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/1916204 (дата обращения: 28.06.2023). – Режим доступа: по подписке.
2. Рейзлин, В. И. Язык С++ и программирование на нём : учебное пособие / В. И. Рейзлин. — 3-е изд., перераб. — Томск : ТПУ, 2021. — 206 с. — ISBN 978-5-4387-0975-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/246239 (дата обращения: 28.06.2023). — Режим доступа: для авториз. пользователей.
3. Ефимова Ю.В. Программирование на языке высокого уровня: Практикум. - Казань: Изд-во Казан. гос. техн. ун-та, 2012. - 32 с.
4. Чукич, И. Функциональное программирование на С++ : учебное пособие / И. Чукич ; перевод с английского В. Ю. Винника, А. Н. Киселева. — Москва : ДМК Пресс, 2020. — 360 с. — ISBN 978-5-97060-781-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/140597 (дата обращения: 28.06.2023). — Режим доступа: для авториз. пользователей.
5. Программирование на языке Си/А.В.Кузин, Е.В.Чумакова - М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с. - Режим доступа: http://znanium.com/bookread2.php?book=505194
6. Язык Си: кратко и ясно: Учебное пособие / Д.В. Парфенов. - М.: Альфа-М: НИЦ ИНФРА-М, 2014. - 320 с. - Режим доступа: https://znanium.com/read?id=356040
7. Программирование графики на С++. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ»

: ИНФРА-М, 2017. — 517 с. - Режим доступа: http://znanium.com/bookread2.php?book=562914

8. Программирование на языке С++: Учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 512 с. - Режим доступа: http://znanium.com/bookread2.php?book=244875