1 Лабораторная работа. Работа с символьной информацией

Цель: получить практические навыки в работе с массивами и указателями языка С, научиться обеспечивать функциональную модульность.

1.1 Задание к лабораторной работе

Согласно выбранному варианту (таблица 1.1), создать функцию для обработки символьных строк. За образец следует взять библиотечные функции обработки строк языка С, но применять их в своей функции не разрешается. Следует предусмотреть обработку ошибок в задании параметров и особые случаи. Требуется разработать два варианта заданной функции, используя традиционную обработку массивов и используя адресную арифметику.

Таблица 1.1 – Варианты заданий

№	Функция	Назначение
1	Copies(s,s1,n)	копирование строки s в строку s1 n раз
2	Words(s)	подсчет слов в строке s
3	Parse(s,t)	разделение строки s на две части: до первого вхождения символа t и после него
4	Center(s1,s2,l)	размещение строки s1 в середине строки s2 длиной l
5	Delete(s,n,l)	удаление из строки s подстроки, начиная с позиции n, длиной l
6	Insert(s,s1,n)	вставка в строку s подстроки s1, начиная с позиции n
7	Reverse(s)	изменение порядка символов в строке s на обратный
8	Pos(s,s1)	поиск первого вхождения подстроки s1 в строку s
9	LastPos(s,s1)	поиск последнего вхождения подстроки s1 в строку s
10	WordIndex(s,n)	определение позиции начала в строке s слова с номером n
11	WordLength(s,n)	определение длины слова с номером n в строке s
12	SubWord(s,n,m)	выделение из строки s m слов, начиная со слова n
13	WordCmp(s1,s2)	сравнение строк (с игнорированием пробелов)
14	Compul(s1,s2)	сравнение строк s1 и s2, игнорируя различия в регистрах
15	Overlay(s,s1,n)	перекрытие части строки s, начиная с позиции n, строкой s1
16	StrSet(s,n,l,t)	установка k символов строки s, начиная с позиции n, в значение t
17	Space(s,l)	доведение строки s до длины l путем равномерной вставки пробелов между словами
18	CopyStr(s1,s2)	копирование подстроки str строки s1 в строку s2, начиная с позиции n

Продолжение таблицы 1.1

19	Findwords(s,s1)	поиск вхождения в строку s заданной фразы s1
20	Replace(s,s1,s2)	замена в строке s подстроки символов s1 на s2
21	Word1(s, s1)	выделение указанного слова s1 из строки s
Примечание - под *«словом»* везде понимается последовательность символов, которая не содержит пробелов.		

1.2 Общие указания к выполнению лабораторной работы

При выполнении задания следует придерживаться следующих рекомендаций:

а) выполнить постановку задачи, с учетом выбранного варианта;

б) при работе со строкой как с массивом нужно иметь в виду, что длина строки заранее неизвестна, поэтому циклы целесообразно организовывать не со счетчиком, а до появления признака конца строки;

в) создаваемая функция *function* должна реализовывать только поставленную задачу — и ничего более. Тогда при ошибочном задании параметров или при каких-то особых случаях в их значениях, функция не должна аварийно завершать программу или выводить какие-то сообщения на экран, но должна возвращать какое-то прогнозируемое значение, по которому можно сделать вывод об ошибке или об особом случае;

г) определить состав параметров функции *function* и установить ее возможные возвращаемые значения;

д) интерпретировать конфигурацию параметров (ограничения, условия) и выбор реакции на неправильное задание;

е) описать логическую структуру программы;

ж) разработать два варианта заданной функции *function***_mas** и *function***_ptr**, используя традиционную обработку массивов и используя адресную арифметику;

з) выполнить реализацию программы и тестирование ее работы. Тестирование должно обеспечить проверку работоспособности функций для всех вариантов входных данных. Входные данные, на которых проводится тестирование, сводятся в таблицу (таблица 1.2).

Пример реализации приведен в приложении А.

Таблица 1.2 – Данные для тестирования

№ теста	Входные данные		Выходные данные	
	…	…	…	…
1				
2				

…				

1.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- листинги программ с комментариями;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

1.4 Контрольные вопросы

1.4.1 В чем заключается специфика системного программного обеспечения?

1.4.2 Какие особенности языка С позволяют использовать его в качестве инструмента для системного программирования?

1.4.3 Каким образом представляются строки символов в С?

1.4.4 Что представляют собой функции?

1.4.5 С какой целью используются прототипы функций?

1.4.6 В чем разница между локальными и глобальными переменными?

1.4.7 Что такое перегрузка функции?

1.4.8 Что представляют собой указатели?

1.4.9 В чем разница между адресом, хранимым в указателе, и значением, записанным по этому адресу?

1.4.10 В чем различие между ссылкой и указателем?

2 Лабораторная работа. Представление в памяти массивов и матриц

Цель: получить практические навыки в использовании указателей и динамических объектов в языке С, а также создании модульных программ и обеспечение инкапсуляции.

2.1 Задание к лабораторной работе

Сформировать разреженную матрицу целых чисел в соответствии с выбранным вариантом задания (см. таблицу 2.1) и создать модуль доступа к ней, в котором следует обеспечить экономию памяти при размещении данных. Способ индексации выбрать самостоятельно.

Таблица 2.1 – Варианты заданий

№	Назначение
1	Все нулевые элементы размещены на главной диагонали, в первых 3 строках выше диагонали и в последних 3 строках ниже диагонали.
2	Все нулевые элементы размещены на местах с нечетными индексами

	строк и столбцов.

Продолжение таблицы 2.1

3	Все нулевые элементы размещены в правой части матрицы.
4	Все нулевые элементы размещены в левой и верхней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти).
5	Все нулевые элементы размещены на местах с четными индексами строк и столбцов.
6	Все нулевые элементы размещены в верхней и нижней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти).
7	Матрица поделена диагоналями на 4 треугольника, элементы верхнего и нижнего треугольников - нулевые.
8	Все нулевые элементы размещены в левой части матрицы.
9	Все нулевые элементы размещены в шахматном порядке, начиная со 2-го элемента 1-й строки.
10	Все нулевые элементы размещены на главной диагонали и в верхней половине участка выше диагонали.
11	Все нулевые элементы размещены в шахматном порядке, начиная с 1-го элемента 1-й строки.
12	Все нулевые элементы размещены в столбцах, индексы которых кратны четырем.
13	Все нулевые элементы размещены на главной диагонали и в нижней половине участка ниже диагонали.
14	Матрица поделена диагоналями на 4 треугольника, элементы левого и правого треугольников нулевые.
15	Матрица поделена диагоналями на 4 треугольника, элементы правого и нижнего треугольников нулевые.
16	Все нулевые элементы размещены попарно в шахматном порядке (сначала 2 нулевых).
17	Все нулевые элементы размещены выше главной диагонали в нечетных строках и ниже главной диагонали — в четных.
18	Все нулевые элементы размещены ниже главной диагонали в нечетных строках и выше главной диагонали — в четных
19	Все нулевые элементы размещены в левой и правой четвертях матрицы (главная и побочная диагонали делят матрицу на четверти).
20	Все нулевые элементы размещены квадратами 2x2 в шахматном порядке.

2.2 Общие указания к выполнению лабораторной работы

При выполнении задания следует придерживаться следующих рекомендаций:

а) выполнить постановку задачи, с учетом выбранного варианта;

б) экономное использование памяти предусматривает, что для тех элементов матрицы, в которых наверняка содержатся нули, память выделяться не будет. Поскольку при этом нарушается двумерная структура матрицы, она может быть представлена в памяти как одномерный массив, но при обращении к элементам матрицы пользователь имеет возможность обращаться к элементу по двум индексам;

в) программное изделие должно быть отдельным модулем - файл *lab2.c*, в котором должны размещаться как данные (матрица и вспомогательная информация), так и функции, которые обеспечивают доступ. Внешний доступ к программам и данным модуля возможен только через вызов функций чтения и записи элементов матрицы. Доступные извне элементы программного модуля должны быть описаны в отдельном файле *lab2.h*, который может включаться в программу пользователя оператором препроцессора: *#include* <*lab2.h*>. Пользователю должен поставляться результат компиляции — файлы *lab2.obj* и *lab2.h*;

г) преобразование 2-компонентного адреса элемента матрицы, которую задает пользователь, в 1-компонентную должно выполняться отдельной функцией (функцией линеаризации), вызов которой возможен только из функций модуля. При этом возможны три метода преобразования адреса:

1) при создании матрицы для нее создается также и дескриптор *D[N]* — отдельный массив, каждый элемент которого соответствует одной строке матрицы; дескриптор заполняется значениями, подобранными так, чтобы

$$n = D[x] + y,$$

где *x, y* — координаты пользователя (строка, столбец), *n* — линейная координата;

2) линейная координата подсчитывается методом итерации как сумма полезных длин всех строк, предшествующих строке *x*, и к ней прибавляется смещение **y**-го полезного элемента относительно начала строки;

3) для преобразования подбирается единое арифметическое выражение, которое реализует функцию: *n = f(x,y).*

Первый вариант обеспечивает быстрейший доступ к элементу матрицы, ибо требует наименьших расчетов при каждом доступе, но плата за это — дополнительные затраты памяти на дескриптор. Второй вариант — наихудший по всем показателям, ибо каждый доступ требует выполнения оператора цикла, а это и медленно, и занимает память. Третий вариант может быть компромиссом, он не требует дополнительной памяти и работает быстрее, чем второй. Но выражение для линеаризации будет сложнее, чем в первом варианте, следовательно, и вычисляться будет медленнее. В программном примере, который приводится в приложении Б, полностью реализован именно третий вариант, но существенные фрагменты программного кода для реализации двух других показаны отдельно;

д) выполнить описание логической структуры программы;

е) для проверки функционирования модуля создается программный модуль, который имитирует программу пользователя.

2.3 Требования к отчету

Отчет по работе выполняется на бумажном носителе и должен содержать:

- задание к работе;
- описание действий, выполненных для получения результата;
- листинги программ с комментариями;
- снимки экрана с результатами работы;
- выводы по каждому заданию.

2.4 Контрольные вопросы

2.4.1 Что представляет собой программный модуль?
2.4.2 Какие переменные являются статическими?
2.4.3 Что представляет собой массив?
2.4.4 Как осуществляется инициализация массива?
2.4.5 Как объявить массив в динамической памяти?
2.4.6 Как удалить массив из динамической памяти?
2.4.7 Что представляет собой разреженный массив?
2.4.8 Можно ли объединять массивы?
2.4.9 Что представляет собой указатель на массив?
2.4.10 Как используются указатели на функции?

Приложение А

Листинг программы и комментарии к лабораторной работе № 1

```
#include <stdio.h>
#include <conio.h>
#define N 80
/*********************************************************/
int substr_mas(char src[N], char dest[N], int num, int len)
{
int i, j;
/* проверка случая 4*/
if((num<0)||(len<=0))
{
dest[0]=0; return 0;
}
/* выход на символ num*/
for (i=0; i<=num; i++)
/* проверка случая 3*/
if(src[i]=='\0')
{
dest[0]=0; return 0;
}
/* перезапись символов*/
for (i--, j=0; j<len; j++, i++)
{
dest[j]=src[i];
/* проверка случая 2*/
if (dest[j]=='\0') return 1;
}
/* запись признака конца в выходную строку*/
dest[j]='\0';
return 1;
}
/*********************************************************/
/* функция выделения подстроки*/
/* адресная арифметика*/
/*********************************************************/
int substr_ptr(char *src, char *dest, int num, int len)
{
/* проверка случая 4*/
if((num<0)||(len<=0)) return dest[0]=0;
/* выход на символ num или на конец строки*/
while (num-- && *src++);
```

```
/* проверка случая 3*/
if(!num) return dest[0]=0;
/* перезапись символов*/
while(len-- && *src) *dest++=*src++;
/* запись признака конца в выходную строку*/
*dest=0;
return 1;
}
/*********************************************************/
main()
{
char ss[N], dd[N];
int n,l;
clrscr();
printf("Enter your symbols:\n");
gets(ss);
printf("begin=");
scanf("%d",&n);
printf("length=");
scanf("%d",&l);
printf("Arrays:\n");
if(substr_mas(ss,dd,n,l)) printf(">>%s<<\n>>%s<<\n",ss,dd);
else printf("Error! >>%s<<\n",dd);
printf("Address arithmetic:\n");
if(substr_ptr(ss,dd,n,l)) printf(">>%s<<\n>>%s<<\n",ss,dd);
else printf("Error! >>%s<<\n",dd);
getch();
}
```

Приложение Б
Листинг программы и комментарии к лабораторной работе № 2

```
/************************ файл lab2.h ***********************/
/* Описание функций и внешних переменных файла lab2.c*/
extern int L2_RESULT;
/* Глобальная переменная – флаг ошибки*/
/* Выделение памяти под матрицу*/
int creat_matr (int N);
/* Чтение элемента матрицы по заданным координатам*/
int read_matr(int x, int y);
/* Запись элемента в матрицу по заданным координатам*/
int write_matr(int x, int y, int value);
/* Уничтожение матрицы*/
int close_matr(void);
/************************ конец файла lab2.h ********************/

/************************** файл lab2.c ***********************/
/* В файле определены функции и переменные для обработки матрицы,
   заполненной нулями ниже главной диагонали*/
#include <alloc.h>
static int NN;                  /* размерность матрицы*/
static int SIZE;                /* размер памяти*/
static int *m_addr=NULL;        /* адрес сжатой матрицы*/
static int lin(int,int);        /* описание функции линеаризации*/
static char ch_coord(int, int); /* описание функции проверки*/
int L2_RESULT;                  /* внешняя переменная – флаг ошибки*/
/************************************************************/
/* Выделение памяти под сжатую матрицу*/
int creat_matr(int N)
/* N – размер матрицы*/
{
NN=N;
SIZE=N*(N-1)/2+N;
if((m_addr=(int *)malloc(SIZE*sizeof(int))) == NULL)
return L2_RESULT=-1;
else
return L2_RESULT=0;
/* Возвращает 0, если выделение памяти прошло успешно, иначе -1 */
}
/************************************************************/
/* Уничтожение матрицы (освобождение памяти*/
int close_matr(void)
{ if (m_addr!=NULL)
```

```
{
free(m_addr);
m_addr=NULL;
return L2_RESULT=0;
/* Возвращает 0, если освобождение памяти прошло успешно, иначе -1*/
}
else return L2_RESULT=-1;
}
/*************************************************************/
/* Чтение элемента матрицы по заданным координатам*/
int read_matr(int x, int y)
/* x, y – координаты (строка, столбец)*/
{
if(ch_coord(x,y)) return 0;
/* Если координаты попадают в нулевой участок, возвращается 0,
иначе - применяется функция линеаризации */
return(x>y) ? 0 : m_addr[lin(x,y)];
/* Проверка успешности чтения по переменной L2_RESULT:
0 – без ошибок, -1 – ошибка  */
}
/*************************************************************/
/* Запись элемента матрицы по заданным координатам*/
int write_matr(int x,int y,int value)
/* x, y – координаты, value – записываемое значение*/
{
if(ch_coord(x,y)) return 0;
/* Если координаты попадают в нулевой участок, записи нет,
 иначе - применяется функция линеаризации*/
if(x>y) return 0;
else return m_addr[lin(x,y)]=value;
/* Проверка успешности записи по переменной L2_RESULT */
}
/*************************************************************/
/*Преобразование 2-мерных координат в линейные (вариант 3)*/
static int lin(int x,int y)
{
int n;
n=NN-x;
return SIZE-n*(n-1)/2-n+y-x;
}
/*************************************************************/
/* Проверка корректности обращения*/
static char ch_coord(int x,int y)
{
```

```
if((m_addr==NULL) || (x>SIZE) || (y>SIZE) || (x<0) || (y<0))
/* Если матрица не размещена в памяти, или заданные координаты
выходят за пределы матрицы */
return L2_RESULT=-1;
return L2_RESULT=0;
}
/********************** конец файла lab2.c *********************/

/************************** файл main2.c ***********************/
/*Программа пользователя*/
#include " lab2.h"
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
main()
{int R,i,j,m;        /*размерность, номера строки и столбца,
                     значения элемента*/
int op;              /* операция*/
clrscr();
printf("Введите размерность матрицы >");
scanf("%d",R);
/* Создание матрицы*/
if (creat_matr(R))
{ printf("Ошибка создания матрицы \n");
exit(0);}
/* Заполнение матрицы*/
for(m=j=0; j<R; j++)
for(i=0; i<R; i++)
write_matr(i,j,++m);
while(1)
/* Вывод матрицы на экран*/
{ clrscr();
for(j=0; j<R; j++)
{ for(i=0; i<R; i++)
printf("%3d",read_matr(i,j));
printf("\n");}
printf("0 – выход \n1 – чтение \n2 - запись\n>");
scanf("%d",&op);
switch (op)
{ case 0:
if(close_matr()) printf("Ошибка при уничтожении\n");
else printf("Матрица уничтожена \n");
exit (0);
case 1: case 2:
```

```
printf("Введите номер строки>");
scanf("%d",&j);
printf("Введите номер столбца>");
scanf("%d",&i);
if (op==2)
{printf("Введите значение элемента>");
scanf("%d",&m);
write_matr(j,i,m);
if(L2_RESULT<0) printf("Ошибка записи\n");}
else
{ m=read_matr(j,i);
if(L2_RESULT<0) printf("Ошибка записи\n");
else printf("Считано: %d\n",m);}
printf("Нажмите клавишу\n");
getch();
break;
}
}
}
```
/********** конец файла main2.c **********/

Вариант, который обеспечивает быстрейший доступ к элементу матрицы, однако требующий дополнительных затрат памяти на дескриптор, реализуется при следующих условиях:

- к общим статическим переменным добавляется переменная

```
static int *D;   /* адрес дескриптора*/
```

- в функцию **create_matr** добавляется блок

```
{int i, s;
D=(int *)malloc(N*sizeof(int));
for (D[0]=0, s=NN-1, i=1; i<NN; i++)
D[i]=D[i-1]+s--;}
```

- функция **lin** изменяется на функцию вида

```
static int lin(int x, int y)
{return D[x]+y;}
```

Наихудший по всем показателям вариант, при котором каждый доступ требует выполнения оператора цикла, что значительно замедляет программу и занимает больше памяти, реализуется при изменении функции **lin** на функцию вида

```
static int lin(int x, int y)
{int s;
for (s=j=0; j<x; j++)
s+=NN-j;
return s+y-x;}
```

Список литературы

1 Молчанов А.Ю. Системное программное обеспечение. Лабораторный практикум. - СПб., 2011.

2 Гордеев В.А. Операционные системы. – СПб.: Питер, 2011.

3 Фельдман С.К. Системное программирование на персональном компьютере. - М.: ЗАО «Новый издательский мир», 2007.

4 Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. – СПб.: Питер, 2008.

5 Финогенов К.Г. Самоучитель по системным функциям MS-DOS. - М.: М., Горячая линия - Телеком, 2001.

6 Страуструп Б. Язык программирования С++. – М.: 2012.

7 Павловская Т.А. С/С++. Структурное программирование. – СПб., 2010.

8 Немцова Т.И. Программирование на языке высокого уровня. Программирование на языке С++. М.: «Форум», 2012.

Содержание