**Лабораторная работа N 2  
ПРЕДСТАВЛЕНИЕ В ПАМЯТИ МАССИВОВ И МАТРИЦ**

**1. Цель работы**

Получение практических навыков в использовании указателей и динамических объектов в языке C, создание модульных программ и обеспечение инкапсуляции.

**2. Темы для предварительного изучения**

* Указатели в языке C.
* Модульная структура программы.
* Области действия имен.

**3. Постановка задачи**

Для разряженной матрицы целых чисел в соответствии с индивидуальным заданием создать модуль доступа к ней, у котором обеспечить экономию памяти при размещении данных.

**4. Порядок выполнения**

Порядок выполнения работы и содержание отчета определены в общих указаниях.

**5. Индивидуальные задания**

|  |  |
| --- | --- |
| **╪п/п** | **Вид матрицы** |
| 1 | все нулевые элементы размещены в левой части матрицы |
| 2 | все нулевые элементы размещены в правой части матрицы |
| 3 | все нулевые элементы размещены выше главной диагонали |
| 4 | все нулевые элементы размещены в верхней части матрицы |
| 5 | все нулевые элементы размещены в нижней части матрицы |
| 6 | все элементы нечетных строк - нулевые |
| 7 | все элементы четных строк - нулевые |
| 8 | все элементы нечетных столбцов - нулевые |
| 9 | все элементы четных столбцов - нулевые |
| 10 | все нулевые элементы размещены в шахматном порядке, начиная с 1-го элемента 1-й строки |
| 11 | все нулевые элементы размещены в шахматном порядке, начиная со-2го элемента 1-й строки |
| 12 | все нулевые элементы размещены на местах с четными индексами строк и столбцов |
| 13 | все нулевые элементы размещены на местах с нечетными индексами строк и столбцов |
| 14 | все нулевые элементы размещены выше главной диагонали на нечетных строках и ниже главной диагонали - на четных |
| 15 | все нулевые элементы размещены ниже главной диагонали на нечетных строках и выше главной диагонали - на четных |
| 16 | все нулевые элементы размещены на главной диагонали, в первых 3 строках выше диагонали и в последних 3 строках ниже диагонали |
| 17 | все нулевые элементы размещены на главной диагонали и в верхней половине участка выше диагонали |
| 18 | все нулевые элементы размещены на главной диагонали и в нижней половине участка ниже диагонали |
| 19 | все нулевые элементы размещены в верхней и нижней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти) |
| 20 | все нулевые элементы размещены в левой и правой четвертях матрицы (главная и побочная диагонали делят матрицу на четверти) |
| 21 | все нулевые элементы размещены в левой и верхней четвертях матрицы (главная и побочная диагонали делят матрицу на четверти) |
| 22 | все нулевые элементы размещены на строках, индексы которых кратны 3 |
| 23 | все нулевые элементы размещены на столбцах, индексы которых кратны 3 |
| 24 | Написать программу, выводящую на экран строку матрицы, сумма элементов которая максимальна |
| 25 | все нулевые элементы размещены на столбцах, индексы которых кратны 4 |
| 26 | все нулевые элементы размещены попарно в шахматном порядке (сначала 2 нулевых) |
| 27 | матрица поделена диагоналями на 4 треугольники, элементы верхнего и нижнего треугольников нулевые |
| 28 | матрица поделена диагоналями на 4 треугольники, элементы левого и правого треугольников нулевые |
| 29 | матрица поделена диагоналями на 4 треугольника, элементы правого и нижнего треугольников нулевые |
|  | 30 все нулевые элементы размещены квадратами 2х2 в шахматном порядке |

Исполнителю самому надлежит выбрать, будут ли начинаться индексы в матрице с 0 или с 1.

**6. Пример решения задачи**

**6.1. Индивидуальное задание:**

* матрица содержит нули ниже главной диагонали;
* индексация начинается с 0.

**6.2. Описание методов решения**

**6.2.1.** Представление в памяти

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.

**6.2.2.** Модульная структура программного изделия

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 - файл LAB2.OBJ и файл LAB2.H. **6.2.3.** Преобразование 2-компонентного адреса элемента матрицы, которую задает пользователь, в 1-компонентную должно выполняться отдельной функцией (так называемой, функцией линеаризации), вызов которой возможен только из функций модуля. Возможны три метода преобразования адреса:

* 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;
* линейная координата подсчитывается методом итерации як сумма полезных длин всех строк, предшествующих строке x, и к ней прибавляется смещение y-го полезного элемента относительно начала строки;
* для преобразования подбирается единое арифметическое выражение, которой 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 тут будет сложнее, чем первом варианте, следовательно, и вычисляться будет медленнее.

В программном примере, который мы приводим ниже, полностью реализован именно третий вариант, но далее мы показываем и существенные фрагменты программного кода для реализации и двух других.

**6.3. Описание логической структуры**

**6.3.1.** Общие переменные

В файле LAB2.C описаны такие статические переменные:

* int NN - размерность матрицы;
* int SIZE - количество ненулевых элементов в матрице;
* int \*m\_addr - адрес сжатой матрицы в памяти, начальное значение этой переменной - NULL - признак того, что память не выделена;
* int L2\_RESULT - общий флаг ошибки, если после выполнения любой функции он равен -1, то произошла ошибка.

Переменные SIZE и m\_addr описаны вне функций с квалификатором static, это означает, что вони доступны для всех функций в этом модуле, но недоступны для внешних модулей. Переменная L2\_RESULT также описана вне всех функций, не без явного квалификатора. Эта переменная доступна не только для этого модуля, но и для всех внешних модулей, если она в них буде описана с квалификатором extern. Такое описание имеется в файле LAB2.H.

**6.3.2.** Функция creat\_matr

Функция creat\_matr предназначена для выделения в динамической памяти места для размещения сжатой матрицы. Прототип функции:

int creat\_matr ( int N );

где N - размерность матрицы.

Функция сохраняет значение параметра в собственной статической переменной и подсчитывает необходимый размер памяти для размещения ненулевых элементов матрицы. Для выделения памяти используется библиотечная функция C malloc. Функция возвращает -1, если при выделении произошла ошибка, или 0, если выделение прошло нормально. При этом переменной L2\_RESULT также присваивается значение 0 или -1.

**6.3.3.** Функция close\_matr

Функция close\_matr предназначена для освобождения памяти при завершении работы с матрицей, Прототип функции:

int close\_matr ( void );

Функция возвращает 0 при успешном освобождении, -1 - при попытке освободить невыделенную память.

Если адрес матрицы в памяти имеет значения NULL, это признак того, что память не выделялась, тогда функция возвращает -1, иначе - освобождает память при помощи библиотечной функции free и записывает адрес матрицы - NULL. Соответственно функция также устанавливает глобальный признак ошибки - L2\_RESULT.

**6.3.4.** Функция read\_matr

Функция read\_matr предназначена для чтения элемента матрицы. Прототип функции:

int read\_matr(int x, int y);

где x и y - координаты (строка и столбец). Функция возвращает значение соответствующего элемента матрицы. Если после выполнения функции значение переменной L2\_RESULT -1, то это указывает на ошибку при обращении.

Проверка корректности задания координат выполняется обращением к функции ch\_coord, если эта последняя возвращает ненулевое значение, выполнение read\_matr на этом и заканчивается. Если же координаты заданы верно, то проверяется попадание заданного элемента в нулевой или ненулевой участок. Элемент находится в нулевом участке, если для него номер строки больше, чем номер столбца. Если элемент в нулевом участке, функция просто возвращает 0, иначе - вызывает функцию линеаризации lin и использует значение, которое возвращает lin, как индекс в массиве m\_addr, по которому и выбирает то значения, которое возвращается.

**6.3.5.** Функция write\_matr

Функция write\_matr предназначена для записи элемента в матрицу. Прототип функции:

int write\_matr(int x, int y, int value);

где x и y - координаты (строка и столбец), value - то значение, которое нужно записать. Функция возвращает значение параметра value, или 0 - если была попытка записи в нулевой участок. Если после выполнения функции значение переменной L2\_RESULT -1, то это указывает на ошибку при обращении.

Выполнение функции подобно функции read\_matr с тем отличием, что, если координаты указывают на ненулевой участок, то функция записывает value в массив m\_addr.

**6.3.6.** Функция ch\_coord

Функция ch\_coord предназначена для проверки корректности задания координат. Эта функция описана как static и поэтому может вызываться только из этого же модуля. Прототип функции:

static char ch\_coord(int x, int y);

где x и y - координаты (строка и столбец). Функция возвращает 0, если координаты верные, -1 - если неверные. Соответственно, функция также устанавливает значение глобальной переменной L2\_RESULT.

Выполнение функции собственно состоит из проверки трех условий:

* адрес матрицы не должен быть NULL, т.е., матрица должна уже находиться в памяти;
* ни одна из координат не может быть меньше 0;
* ни одна из координат не может быть больше NN.

Если хотя бы одно из этих условий не выполняется, функция устанавливает признак ошибки.

**6.3.7.** Функция lin

Функция lin предназначена для преобразования двумерных координат в индекс в одномерном массиве. Эта функция описана как static и поэтому может вызываться только из этого же модуля. Прототип функции:

static int lin(int x, int y);

где x и y - координаты (строка и столбец). Функция возвращает координату в массиве m\_addr.

Выражение, значение которого вычисляет и возвращает функция, подобрано вот из каких соображений. Пусть мы имеет такую матрицу, как показано ниже, и нам нужно найти линейную координату элемента, обозначенного буквой A с координатами (x,y):

x x x x x x

0 x x x x x

0 0 x x A x

0 0 0 x x x

0 0 0 0 x x

0 0 0 0 0 x

Координату элемента можно определить как:

n = SIZE - sizeX +offY,

где SIZE - общее количество элементов в матрице (см. creat\_matr),

SIZE = NN \* (NN - 1) / 2 + NN;

sizeX - количество ненулевых элементов, которые содержатся в строке x и ниже,

sizeX = (NN - x) \* (NN - x - 1) / 2 + (NN - x);

offY - смещение нужного элемента от начала строки x,

offY = y - x.

**6.4. Программа пользователя**

Для проверки функционирования нашего модуля создается программный модуль, который имитирует программу пользователя. Этот модуль обращается к функции creat\_matr для создания матрицы нужного размера, заполняет ненулевую ее часть последовательно увеличивающимися числами, используя для этого функцию write\_matr, и выводит матрицу на экран, используя для выборки ее элементов функцию read\_matr. Далее в диалоговом режиме программа вводит запрос на свои действия и читает/пишет элементы матрицы с заданными координатами, обращаясь к функциям read\_matr/write\_matr. Если пользователь захотел закончить работу, программа вызывает функцию close\_matr.

**6.5. Тексты программных модулей**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Файл LAB2.H \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Описание функций и внешних переменных файла LAB2.C \*/

extern int L2\_RESULT; /\* Глобальна переменна - 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 элемент 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 этом 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 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;

}

else return L2\_RESULT=-1;

/\* Возвращает 0, если освобождение пршло успешно, иначе - -1 \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Чтение элемента матрицы по заданным координатам \*/

int read\_matr(int x, int y) {

/\* x, y -координати (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 - была ошибка \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Запись элемента матрицы по заданным координатам \*/

int write\_matr(int x, int y, int value) {

/\* x, y -координати, value - записываемое значение \*/

if ( chcoord(x,y) ) return;

/\* 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 L2\_RESULT \*/

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Преобразование 2-мерних координат в линейную \*/

/\* (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"

main(){

int R; /\* размерность \*/

int i, j; /\* номера строки и столбца \*/

int m; /\* 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=о; 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) pritnf("Ошибка записи\n");

}

else {

m=read\_matr(j,i);

if (L2\_RESULT<0) pritnf("Ошибка считывания\n");

else printf("Считано: %d\n",m);

}

printf("Нажмите клавишу\n"); getch();

break;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Конец файла MAIN2.C \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**6.6. Варианты.**

Ниже приведены фрагменты программных кодов, которые отличают варианты, рассмотренные в 6.2.3.

Вариант 1 требует:

* добавления к общим статическим переменным еще переменной:
* static int \*D; /\* адрес дескриптора \*/
* добавления такого блока в функцию creat\_matr:
* {
* 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--;
* }
* изменения функции lin на:
* static int lin(int x, int y) {
* return D[x]+y;
* }

Вариант 2 требует:

* изменения функции lin на:
* static int lin(int x, int y) {
* int s;
* for (s=j=0; j<x; j++)
* s+=NN-j;
* return s+y-x;

}