**Лабораторная работа №4**

**Работа с указателями. Передача параметров в функцию через указатель и ссылку. Обработка числовых массивов.**

***Цель работы:***

1. Изучить методы работы с указателями.
2. Продолжить изучение использования пользовательских функций:

* Применить указатели при взаимодействии с функциями (передача параметров и возврат значений)
* Применить указатели и функции в задачах по обработке числовых массивов.
* Применить указатели и функции в задачах обработки текстов.

1. Применить библиотечные функции в задачах обработки текстов.

**Теоретические сведения.**

**Указатели.**

Работая с памятью, программист оперирует понятиями имя и значение переменной – это работа на высоком уровне абстракции. На машинном (низком уровне), компьютер оперирует понятиями «адрес памяти» и «содержимое памяти». Указатели позволяют программисту работать на низком уровне абстракции, оперируя адресами, а не именами объектов.

Определяя переменную, например: int var =1; программа заставляет компилятор выделять память, при этом имя переменной var адресует участок памяти, а константа 1 определяет значение, которое запишется по этому адресу.

Имея в своем распоряжении адрес переменной, программисту необходимо с ним работать: сохранять, передавать и преобразовывать, для этой цели и служат указатели.

*Указатели - это переменные, значениями которых являются адреса других объектов - переменных, массивов, функций*.

*Указатели служат для обращения к области памяти, отведенной под другую переменную.*

Итак, для того, чтобы работать с указателем, необходимы, по крайней мере, две переменные – сам указатель и переменная на которую он будет ссылаться. Подобно любой другой переменной, указатель нужно создать (объявить) и задать начальное значение (установить указатель), после чего он будет указывать (ссылаться) на переменную. При объявлении указателя перед именем ставится знак \*.

**Пример:** Объявление и установка указателей

int x, \*p\_i,; // объявить (выделить память) переменную x и указатель p\_i

double f, \*p\_d; // объявить переменную f и указатель p\_d

p\_i = &x ; // установить указатель p\_i на переменную x

p\_d = &f ;

Обратите внимание, что указатель должен быть того же типа, что и переменная, на которую он ссылается.

Операция &x означает: "получить адрес переменной x".

Оператор p\_d = &f означает следующее: адрес переменной f занести в указатель p\_d (указатель установлен на переменную f).

Установка указателя на объект – это обязательный этап работы с указателем. Будьте внимательны, неустановленный указатель главный источник неприятностей!

После того, как указатель установлен, можно обращаться к объекту, на который он указывает, для этой цели служит специальная операция – разыменование «\*». Операция \* рассматривает свой операнд как адрес и позволяет обратиться к содержимому этого адреса.

**Пример:** Использованиеуказателей в выражениях.

int x, \*ptr;

ptr = &x; // ptr ссылается на x

\*ptr= \*ptr +1; // аналог: x = x+1

\*ptr = \*ptr\*5; // x=x\*5

**Указатели и массивы.**

В языке С между указателями и массивами существует тесная связь.

Рассмотрим, что происходит, когда объявляется массив, например int array[25]:

* + выделяется память для двадцати пяти элементов массива типа int
  + создается указатель на начало выделенной памяти c именем **array**

Массив остается безымянным, а доступ к его элементам осуществляется через указатель array. Следует отметить, что указатель array является константным, его значение нельзя менять, то есть он всегда указывает на нулевой элемент массива (его начало).

Имя массива можно приравнять к указателю, так как оно также является указателем.

int arrаy[25], \*ptr;

ptr = array; // установка указателя ptr на начало массива

Оператор **ptr=array** можно заменить на эквивалентный ему: **ptr=&array[0].**

К элементу массива можно обратиться двумя способами:

* с помощью индекса
* с помощью указателя.

**Пример:** Сравнение двух способов доступа к элементам массива.

// определение массива **mas** и указателя **q**

char mas[100] , \*q;

q = mas; // установка указателя на начало массива

// Доступ к элементам массива (следующие записи эквивалентны):

mas[0]; // \*(q+0)

mas[10]; // \*(q+10)

mas[n-1]; // \*(q+n-1)

**Арифметика указателей.**

Работая с указателем программист имеет возможность совершать действия двух различных типов:

* обращаться к объекту (к данным), на который ссылается указатель, используя операцию \* (обращение по адресу указателя);
* изменять адрес, записанный в указателе, таким образом, меняя объект с которым указатель связан.

Манипуляции с адресом часто *называют арифметикой указателей или адресной арифметикой.* Это очень удобный инструмент, позволяющий легко манипулировать массивами как простых, так и сложных типов данных. Арифметические операции с адресами имеют ряд ограничений, которые мы и рассмотрим подробнее.

**Присваивание**

Операцию присваивания можно применять :

* при занесении адреса в указатель;
* при занесении нулевого значения в указатель;
* два указателя можно приравнять друг другу.

*Никакие другие типы присваивания с указателями недопустимы.*

**Пример:**

int \*p,\*k,\*z,date=2007;

p=&date; // установка указателя на переменную date

k=p; // значение указателя p заносится в указатель k

z=NULL; // нулевое значение занести в указатель

**Инкремент и декремент**

Операции ++ и -- применённые к указателю перемещают ссылку на следующий (или предыдущий) объект, позволяя последовательно перебирать элементы массива в прямом или обратном направлении.

**Пример:**

float m[10], \*pk;

char q[60],\*ps;

ps=q; // установить указатель на q[0] (первый элемент)

pk=&m[9]; // установить указатель на m[9] (последний элемент)

ps++; // переместить указатель на q[1]

pk--; // переместить указатель на m[8]

Заметим, что в данном примере обращение к элементам массива нет, мы лишь манипулируем с адресами в указателях. Для того, чтобы обратиться к данным, например, напечатать элемент массива, нужно применить операцию разыменования

printf("%c \n",\*ps); // печать q[1]

printf("%f \n",\*pk); // печать m[8]

**Сложение**

*Два указателя нельзя складывать, однако к указателю можно прибавить целую величину.*

Если помнить, что указатели содержат адреса, то подобные ограничения вполне объяснимы, действительно, складывать между собой два разных адреса не имеет смысла, а вот прибавить к адресу некоторое число, означает переместиться вперед, «перешагнув» через несколько объектов.

**Пример: В**ывести каждый четвертый элемент массива на печать.

double mas[N], \*pk;

. . . // инициализация массива

for (pk=&mas[0] ; pk<&mas[N-1]; pk=pk+3)

printf("\t%f",\*pk); // печать элемента массива

**Вычитание**

*Два указателя можно вычитать друг из друга, если они одного типа.*

Эта операция дает расстояние между объектами. Из адреса указателя можно вычесть целую величину, таким образом, перемещаясь в сторону уменьшения адресов.

**Пример:**

Вывести каждый второй элемент массива в обратном порядке.

…

for (pk=&mas[N-1] ; pk>M; pk=pk-1)

printf("\t%f",\*pk); // печать элемента массива

**Указатели в параметрах функции.**

Переменные, объявленные в функциях - это локальные (временные) переменные, которые уничтожаются при выходе из функции, а значит, их значения теряются. Когда же нужно сохранить вычисляемые значения требуется располагать переменные за пределами функции. В других случаях требуется обрабатывать данные, расположенные за пределами функции. Поэтому достаточно часто возникает необходимость обращения к объектам, определенным вне функции, которые являются внешними по отношению к функции.

Для этой цели служат указатели, и это одно из важнейших применений указателя - организация связи функции с внешней средой.

*Указатели в параметрах функции имеют два важных применения:*

* доступ к памяти, находящейся за пределами функции (доступ к «внешней» памяти);
* возврат из функции более одного значения.

*Если функция должна обращаться к внешней памяти, то параметры следует передавать через указатели.*

**Пример:** Функция изменяет переменную в вызывающей программе.

#include <stdio.h>

void inp\_kl (int \*p) // ввод числа с клавиатуры в переменную функции main()

{

printf ("\nx=");

scanf\_s("%d",p); // второй параметр функции scanf – адрес переменной для ввода данных

}

int main()

{

int x=0;

inp\_kl(&x); // вызов функции inp\_kl ()

printf("x=%d\n",x);

}

Функция **inp\_kl()** имеет один параметр – указатель на **int**, который устанавливается при вызове функции (фактическим параметром). То есть при вызове **inp\_kl(&x)** происходит создание и установка указателя p (формального параметра функции) на переменную **x,** то есть происходит следующее действие: **int\* p = &x;**

Таким образом, функция **inp\_kl()** получает доступ к переменной **x** через указатель **p**

Подобным образом могут изменяться и внешние массивы. В функцию передаётся указатель на массив, через который и ведётся работа с его элементами.

**Примеры программирования.**

**Пример 1:** Создать массив с помощью датчика случайных чисел, распечатать на экране и найти его сумму.

#include <stdio.h>

#include <stdlib.h>

#define N 10 // размер массива

#define COL 5 // число колонок при выводе массива на экран

int main()

{// область определений (выделение памяти)

double \*k; // указатель на тип double

double dig[N], s; // массив dig и переменные s и i

int i ;

//-----------------------------------------------------------

// Заполнить массив случайными числами

for (k=dig; k<dig+N; k++)

\*k=rand()%100/10.;

// печать исходного массива

for (k=dig,i=0; k<dig+N; k++, i++)

{

printf("%8.2f",\*k);

if ((i+1)%COL)

printf("\t");

else

printf("\n");

}

// суммирование массива

for (s=0,k=dig; k<dig+N; k++)

s=s+ \*k;

// печать результата

printf ("\n===============\ns=%f\n",s);

return 0;

}

Рассмотрим цикл, заполняющий массив **for (k=dig; k<dig+N; k++)**:

* перед началом цикла указатель устанавливается на начало массива, **k=dig**,
* цикл продолжается до тех пор, пока значение указателя меньше адреса последнего элемента массива, **k<dig+N**,
* на каждом проходе цикла дается приращение к адресу указателя **k++ ,** таким образом, указатель перемещается на следующий элемент

В цикле происходит перебор всех элементов массива от первого до последнего, тело цикла состоит их одного оператора **\*k=rand()%99/3.;**

Указатель **k** ссылается на очередной элемент, то есть содержит адрес этого элемента, а операция **\*k** – обращение к данным этого элемента. Функция **rand()** возвращает случайное число целого типа, путем несложного вычисления преобразуем его в вещественное число. Поэтому оператор **\*k=rand()%99/3.;** не что иное, как запись в текущий элемент массива случайного вещественного числа.

В цикле печати, кроме указателя **k**, адресующего элементы массива, используется счётчик выведенных чисел **i**, который используется для форматирования. Если **i** кратно **COL**, то необходимо закончить строку (вывести на печать символ «перевод строки»).

Суммирование массива проводится в отдельном цикле.

**Пример 2:** Решим ту же задачу с применением функций. Потребуются три функции: для заполнения массива, для вывода массива на экран и для суммирования массива.

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#define N 10 // размер массива

#define COL 5 // число колонок массива на экране

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ область определения функций\_\_\_\_\_\_\_\_

**// функция инициализации массива**

// генерация вещественных положительных чисел

void initmas (double \*p, int n)

{

double \*tp; // рабочий указатель (локальная переменная)

for (tp=p; tp < (p+n); tp++)

\*tp=rand()%100/10.;

}

**// функция печати массива**

void printmas (double \*p, int n, int k)

{

int i;

for (i=0; i<n; i++,p++)

{

printf("%8.2f",\*p); // печать элемента массива

if ((i+1)%k)

printf("\t"); // выбор разделителя

else

printf("\n");

}

printf("\n");

}

**// функция вычисления суммы**

double summas (double \*p, int n)

{

double fs; //сумма (локальная переменная)

double \*tp; // рабочий указатель (локальная переменная)

for (fs=0, tp=p; tp < (p+n); tp++)

fs=fs+ \*tp; // суммирование массива

return fs; // возврат суммы в вызывающую программу

}

//\_\_\_\_\_\_\_\_\_ конец области определения функций\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int main()

{//---------------------------------------- настройки

setlocale(LC\_ALL,"Rus");

srand (time(0));

//------ область определений (выделение памяти)

double dig[N], s;

//----------------------------------------------------------------

initmas(dig,N); // инициализация массива dig

printmas(dig,N,COL); // печать в 5 колонок

s = summas(dig,N); // вычисление суммы

printf ("s=%f\n",s);

return 0;

}

Вся необходимая память (dig[] и s) определена в вызывающей программе, поэтому эта память является внешней, по отношению к трём пользовательским функциям: **initmas(), printmas() и summas()**. Разберем подробно работу с внешним массивом на примере функции **initmas().**

Чтобы функция имела доступ к массиву dig(), необходимо передать указатель на массив (адрес начала) в качестве параметра функции (первый параметр) и длину массива (второй параметр). При вызове функции **initmas(dig, N)** в функции **main(**)) происходит инициализация формальных параметров (**p** и **n)** функции **initmas(),** при входе в функцию выполняются следующие действия :

**int\* p = dig ,int n = N**, то есть создается указатель **p** и связывается с началом массива **dig.** Также создается переменная n и инициализируется длинной массива.

Для текущей работы потребуется ещё один указатель **(tp),** текущая работа заключается в перемещении указателя на очередной элемент массива.

Рассмотрим заголовок цикла **for (tp=p; tp < (p+n); tp++)**

* перед началом цикла текущий указатель устанавливается на начало массива, то есть **tp=p**
* цикл выполняется до тех пор, пока текущий указатель меньше адреса последнего элемента массива (выражение **(p+n)** дает адрес последнего элемента)
* на каждом проходе цикла текущий указатель перемещается на следующий элемент, то есть **tp++**

Теперь понятно, почему потребовался дополнительный указатель **tp,** в самом деле, если на каждом проходе цикла необходимо проверять условие выхода из цикла**,** то следует где-то хранить адрес начала массива для того, чтобы вычислить адреса конца массива (**p+n**), а значит перемещать указатель **p** нельзя. Следует отметить, что когда мы работаем с массивами через указатель, всегда требуется как минимум два указателя – один для хранения «начала отсчета», а другой для текущей работы.

Две другие функции **printmas() и summas()** используют те же принципы работы с указателями.

Обратите внимание, на то, как упростилась функция main(), логика выполнения программы стала хорошо видна, остались только основные переменные. Все «детали работы» и вспомогательные переменные отошли к функциям, или как часто говорят «скрыты в функциях», и это существенно экономит как память компьютера, так и время на разработку программы. В самом деле, стоит вспомнить, что функцию можно многократно вызывать из различных частей программы с различными параметрами. Поэтому однажды написанная функция может многократно использоваться. Следующие примеры показывают, разнообразную работу с функциями.

**Пример 3:**  Поставим задачу формирования массива результатов. Дан исходный массив вещественных чисел, сформировать результирующий массив из отрицательных элементов исходного массива.

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#define N 20 // размер массива

#define COL 5 // число колонок массива на экране

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ область определения функций\_\_\_\_\_\_\_\_

**// функция инициализации массива**

// генерация положительных и отрицательных чисел

void initmas (double \*p, int n)

{

double \*tp; // рабочий указатель

for (tp=p; tp < (p+n); tp++)

\*tp=rand()%1000/10.-50;

}

**// функция печати массива**

void printmas (double \*p, int n, int k)

{

int i;

for (i=0; i<n; i++,p++)

{

printf("%8.2f",\*p); // печать элемента массива

if ((i+1)%k)

printf("\t"); // выбор разделителя

else

printf("\n");

}

printf("\n");

}

**// Функция поиска отрицательных элементов массива**

// формирование массива результатов (данные)

// p - указатель на исходный массив, n - размер исходного массива

// r – указатель на результирующий массив

// возврат значения (размер массива результатов)

int minusmas (double\* p, int n, double\* r)

{

double\* tp; // рабочий указатель для исходного массива

double\* tr; // рабочий указатель для массива результатов

for (tp=p,tr=r; tp < (p+n); tp++)

if (\*tp<0)

{

\*tr=\*tp;

tr++;

}

return (tr-r); // размер массива результатов

}

//\_\_\_\_\_\_\_\_\_ конец области определения функций\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

int main()

{//---------------------------------------- настройки

setlocale(LC\_ALL,"Rus");

srand (time(0));

// область определений (выделение памяти)

double dig[N],rez[N];

//----------------------------------------------------------------------

initmas(dig,N); // инициализация исходного массива

printmas(dig,N,COL); // печать исходного массива

int size=minusmas(dig,N,rez);// формирование массива отрицательных элементов

printf ("Массив результатов ---------------------------------\n");

printmas(rez,size,COL); // печать массива результатов

return 0;

}

Когда нужно сформировать массив результатов, размер которого заранее не известен, можно поступить следующим образом:

* Выделить память заведомо большего размера (с запасом). В нашем случае размер результирующего массива **rez** равен размеру исходного массива **dig**
* В процессе работы вычислить истинный размер результирующего массива. В нашем случае функция **minusmas()** возвращает размер массива с результатами

**Вопросы.**

1. Дайте понятие указателя. Для каких целей он служит?
2. Поясните следующие понятия: установка указателя, тип указателя.
3. Какие действия выполняют операции \* и & ?
4. Какие действия необходимо выполнить, чтобы обратиться к переменной через указатель (начиная с объявления указателя).
5. Что такое адресная арифметика?
6. Какие ограничения действуют в адресной арифметике на операцию «присваивания»?
7. Какие ограничения действуют в адресной арифметике на операции «инкремента» и «декремента»?
8. Какие ограничения действуют в адресной арифметике на «сложение» и «вычитание»?
9. Что представляет собой массив данных с точки зрения указателей?
10. Перечислите три направления применения указателей в параметрах функций.
11. Как функция может обратиться к памяти, находящейся за её пределами? Приведите пример.

|  |  |  |
| --- | --- | --- |
| Варианты задания | Условия задания | |
| 1, 16 | Дана целочисленная матрица NxN  Создать функцию, которая формирует одномерный массив, состоящий из сумм элементов нечетных столбцов. | |
|  | Дана целочисленная матрица NxN  Создать функцию, которая возвращает 2 значения:  минимум и максимум заштрихованной области |
| 2, 17 | Дана целочисленная матрица NxM  Создать функцию, которая формирует одномерный массив, состоящий из сумм элементов четных строк. | |
|  | Дана целочисленная матрица NxM  Создать функцию, которая возвращает 2 значения:  максимум в верхней заштрихованной области и минимум в нижней. |
| 3, 18 | Дана целочисленная матрица NxN  Создать функцию, которая формирует одномерный массив, содержащий количество отрицательных элементов в нечетных столбцах. | |
|  | Дана целочисленная матрица NxN  Создать функцию, которая возвращает 2 значения:  Сумму элементов левой и правой заштрихованной области. |
| 4, 19 | Дана целочисленная матрица NxN  Создать функцию, которая формирует одномерный массив, содержащий средние арифметические значения строк. | |
|  | Дана целочисленная матрица NxN  Создать функцию, которая возвращает 2 значения:  минимум и максимум заштрихованной области. |
| 5, 20 | Дана целочисленная матрица NxM  Создать функцию, которая формирует одномерный массив, состоящий из сумм элементов нечетных столбцов. | |
|  | Дана целочисленная матрица NxM  Создать функцию, которая возвращает 2 значения:  суммы верхней и нижней заштрихованной области. |
| 6, 21 | Дана целочисленная матрица NxM  Создать функцию, которая формирует одномерный массив, содержащий количество нулевых элементов четных столбцов. | |
|  | Дана целочисленная матрица NxM  Создать функцию, которая возвращает 2 значения:  минимум и максимум заштрихованной области. |
| 7, 22 | Дана целочисленная матрица NxN. Создать функцию, которая формирует одномерный массив, содержащий средние арифметические значения каждого столбца. | |
|  | Дана целочисленная матрица NxN.  Создать функцию, которая возвращает 2 значения:  максимальный и минимальный по модулю элемент в заштрихованной области. |
| 8, 23 | Дана целочисленная матрица NxM. Создать функцию, которая формирует одномерный массив, содержащий количество нулей в нечетных строках. | |
|  | Дана целочисленная матрица NxM.  Создать функцию, которая возвращает 2 значения:  максимум в верхней заштрихованной области и минимум в нижней. |
| 9, 24 | Дана целочисленная матрица NxN  Создать функцию, которая формирует одномерный массив, содержащий минимумы (по модулю) каждого столбца. | |
|  | Дана целочисленная матрица NxN  Создать функцию, которая возвращает 2 значения:  Найти количество нулей в верхней заштрихованной области и количество положительных элементов в нижней. |
| 10, 25 | Дана целочисленная матрица NxM , содержащая как положительные, так и отрицательные элементы.  Создать функцию, которая формирует одномерный массив, содержащий количество отрицательных элементов в четных столбцах. | |
|  | Дана целочисленная матрица NxM, содержащая как положительные, так и отрицательные элементы.  Создать функцию, которая возвращает 2 значения:  количество положительных и количество отрицательных элементов в заштрихованной области. |
| 11, 26 | Дана целочисленная матрица NxN, содержащая как положительные, так и отрицательные элементы.  Создать функцию, которая формирует одномерный массив, содержащий максимальные значения четных строк. | |
|  | Дана целочисленная матрица NxN, содержащая как положительные, так и отрицательные элементы.  Создать функцию, которая возвращает 2 значения:  количество положительных четных и отрицательных четных элементов в заштрихованной области. |
| 12, 27 | Дана целочисленная матрица NxN  Создать функцию, которая формирует одномерный массив, содержащий количество нулей в нечетных столбцах. | |
|  | Дана целочисленная матрица NxN  Создать функцию, которая возвращает 2 значения:  минимум в левой заштрихованной области и максимум в правой. |
| 13, 28 | Дана целочисленная матрица NxM  Создать функцию, которая формирует одномерный массив, содержащий среднее арифметическое каждого столбца. | |
|  | Дана целочисленная матрица NxM  Создать функцию, которая возвращает 2 значения:  среднее арифметическое каждой заштрихованной области (отдельно верхней и нижней). |
| 14, 29 | Дана целочисленная матрица NxM, содержащая как положительные, так и отрицательные элементы  Создать функцию, которая формирует одномерный массив, содержащий минимальные по модулю значения нечетных столбцов. | |
|  | Дана целочисленная матрица NxM, содержащая как положительные, так и отрицательные элементы.  Создать функцию, которая возвращает 2 значения:  среднее арифметическое положительных и отрицательных элементов заштрихованной области. |
| 15, 30 | Дана целочисленная матрица NxM.  Создать функцию, которая формирует одномерный массив, содержащий суммы четных строк. | |
|  | Дана целочисленная матрица NxM.  Создать функцию, которая возвращает 2 значения:  количество нулей в каждой заштрихованной области (отдельно верхней и нижней). |