

Лабораторная работа №5

Работа с динамической памятью. Обработка числовых и текстовых данных.

Цель работы:

1. Продолжить изучение методов работы с указателями и пользовательскими функциями.
2. Изучить методы работы с динамической памятью
3. Изучить методы работы с текстовыми данными

Теоретические сведения Динамическое выделение памяти

Память, которую использует программа, делится на три вида:

1. Статическая память (static memory)
 - хранит глобальные переменные и константы;
 - размер определяется при компиляции.
2. Стек (stack)
 - хранит локальные переменные, аргументы функций и промежуточные значения вычислений;
 - размер определяется при запуске программы (обычно выделяется 4 Мб).
3. Куча (heap)
 - динамически распределяемая память;
 - ОС выделяет память по частям (по мере необходимости).

Динамически распределяемую память следует использовать, в случае если мы заранее (на момент написания программы) не знаем, сколько памяти нам понадобится (например, размер массива зависит от того, какие данные введет пользователь во время работы программы) и при работе с большими объемами данных.

Динамическая память, называемая также "кучей", выделяется явно по запросу программы из ресурсов операционной системы и доступна через указатель. Она не инициализируется автоматически при выделении и должна быть явно освобождена после использования. Динамическая память ограничена лишь размером оперативной памяти и может меняться в процессе работы программы

Работа с динамической памятью в С

Для работы с динамической памятью в языке С используются следующие функции: **malloc**, **calloc**, **free**, **realloc**.

Рассмотрим их подробнее.

1. При помощи библиотечной функции **malloc** (**calloc**)

// выделения памяти под 1 000 элементов типа int

```
int * p = (int *) malloc (1000*sizeof(int));
```

```
if (p==NULL) cout<< "\n память не выделена";
```

```
...
```

```
free (p);    // возврат памяти в кучу
```

2. Выделение (захват памяти) : `void *calloc(size_t nmemb, size_t size);`

Функция работает аналогично `malloc`, но отличается синтаксисом (вместо размера выделяемой памяти нужно задать количество элементов и размер одного элемента) и тем, что выделенная память будет обнулена. Например, после выполнения

```
int * p = (int *) calloc(1000, sizeof(int))
```

`p` будет указывать на начало массива типа `int` из 1000 элементов, инициализированных нулями.

3. Изменение размера памяти : `void *realloc(void *ptr, size_t size);`

Функция изменяет размер выделенной памяти (на которую указывает `ptr`, полученный из вызова `malloc`, `calloc` или `realloc`). Если размер, указанный в параметре `size` больше, чем тот, который был выделен под указатель `ptr`, то проверяется, есть ли возможность выделить недостающие ячейки памяти подряд с уже выделенными. Если места недостаточно, то выделяется новый участок памяти размером `size` и данные по указателю `ptr` копируются в начало нового участка.

Практическое применение

В процессе выполнения программы участок динамической памяти доступен везде, где доступен указатель, адресующий этот участок. Таким образом, возможны следующие три варианта работы с динамической памятью, выделяемой в некотором блоке (например, в теле неглавной функции).

1. Указатель (на участок динамической памяти) определен как локальный объект автоматической памяти. В этом случае выделенная память будет недоступна при выходе за пределы блока локализации указателя, и ее нужно освободить перед выходом из блока.

```
void f (int n)
{
    int* p= (int *) calloc(n, sizeof(int))
    ...
    free (p); // освобождение дин. памяти
}
```

2. Указатель определен как локальный объект статической памяти. Динамическая память, выделенная однократно в блоке, доступна через указатель при каждом повторном входе в блок. Память нужно освободить только по окончании ее использования.

```
#include "stdafx.h"
#include <stdlib.h>
#include <iostream>
using namespace std;
// функция с несколькими режимами работы (выделение, работа, освобождение
динамической памяти)
void f1 (int k, int n) // k- режим работы, n-размер массива
{static int* p;
int* tp;
switch (k)
{case 1:{p= (int *) calloc(n, sizeof(int));break;}
case 2:{cout<<"----- "<<n<<" -----"<<endl;
for (tp=p;tp<p+n;tp++) *tp=rand()%100; // инициализация
for (tp=p;tp<p+n;tp++) cout<<*tp<<"\t"; // печать
break;}
case 3:{ cout<<"-----"<<endl;
free (p); break; // освобождение (возврат) дин. памяти
}
};
}
```

```

int _tmain(int argc, _TCHAR* argv[])
{f1(1,10);    //выделение дин. памяти (10 элементов)
f1(2,10);    // инициализация и печать
f1 (3,1);    // освобождение дин. памяти
system ("pause");
f1(1,50);    //выделение дин. памяти (50 элементов)
f1(2,50);
f1 (3,1);
system ("pause");
return 0;
}

```

3. Указатель является глобальным объектом. Динамическая память доступна во всех блоках. Память нужно освободить только по окончании ее использования

```

#include "stdafx.h"
#include <stdlib.h>
#include <iostream>
using namespace std;
#define n 10
int* pG;    //рабочий указатель для дин. памяти (глобальная переменная)
void init (int size)  // функция для инициализации массива
{int i;
for (i=0; i< size; i++) //цикл ввода чисел
{ printf("x[%d]=",i);  scanf("%d", &pG[i]);}
}
//-----
int sum (int size)      // вычисление суммы массива
{int i,s=0;
for (i=0; i< size; i++) //цикл суммирования
s=s+pG[i];
return s;
}

int _tmain(int argc, _TCHAR* argv[])
{
pG= (int *) calloc(n, sizeof(int));    // выделение памяти
init (n);    //работа с дин.памятью
printf("\ns=%d\n",sum(n));
free (pG); pG=NULL;    // освобождение памяти
system ("pause");
return 0;
}

```

Работа с динамической памятью в C++

В C++ есть свой механизм выделения и освобождения памяти — это функции **new** и **delete**.

Пример использования **new**:

```
int * p = new int[1000]; // выделение памяти под 1000 эл-тов
```

Т.е. при использовании функции **new** не нужно приводить указатель и не нужно использовать **sizeof()**.

Освобождение выделенной при помощи **new** памяти осуществляется посредством следующего вызова: **delete [] p;**

Пример:

```
#include "stdafx.h"
#include <stdlib.h>
#include <iostream>
using namespace std;
#define n 10
int _tmain(int argc, _TCHAR* argv[])
{
    int * p = new int[n]; // выделение памяти под n эл-тов
    int *tp;
    for (tp=p;tp<p+n;tp++) *tp=rand()%100; // инициализация
    for (tp=p;tp<p+n;tp++) cout<<*tp<<"\t"; // печать
    delete [] p; // освобождение памяти
    system ("pause");
    return 0;
}
```

Если требуется выделить память под один элемент, то можно использовать

```
int * q = new int; // выделение памяти под переменную типа int
```

или

```
int * q = new int(10); // выделение и инициализация (числом 10) памяти под переменную типа int
```

в этом случае удаление будет выглядеть следующим образом:

```
delete q;
```

Проверка выделения памяти

Существует единственное числовое значение, которое можно присвоить непосредственно указателю — это **NULL**. Нулевой адрес — особый, по этому адресу не может храниться ни одна переменная. То есть указатель, имеющий нулевое значение указывает в "никуда", к такому указателю нельзя применить оператор разыменования.

Библиотечные функции **malloc** (**calloc**) или оператор **new** используют функцию операционной системы для выделения памяти. Если затребованный размер памяти слишком большой (а также при попытке создать массив из нуля или отрицательного числа элементов), операционная система не будет выделять память и тогда функции или оператору вернет нулевое значение (**NULL**).

Пример:

```
int n=1000000000;  
int *pi=new int[n];  
if (pi==NULL) { // if (!pi)  
    printf ("Требуемая память не выделена!");
```

Типичные ошибки при работе с динамической памятью

При работе с динамической памятью можно совершить большое количество ошибок, которые имеют различные последствия и различную степень тяжести.

1. Попытка воспользоваться неинициализированным указателем

```
float *pi;  
*pi=3.14;    //использование неинициализированного указателя
```

Если **pi** – глобальная переменная, то она автоматически инициализируется нулевым значением, т.е. имеет значение **NULL**. Разыменование нулевого указателя приводит к ошибке времени выполнения. Если **pi** – локальная переменная, то она по умолчанию не инициализируется, а поэтому содержит непредсказуемое значение. Это значение трактуется как адрес переменной, к которой осуществляется доступ.

По чистой случайности может оказаться, что указатель **pi** содержит истинный адрес памяти программы, тогда обращение к памяти через этот указатель не вызовет ошибки времени выполнения, использование такого указателя непредсказуемым образом повлияет на дальнейшее выполнение программы.

2. Не освобожденные указатели.

После освобождения динамической памяти указатель продолжает указывать на прежний адрес памяти. Такие указатели представляют потенциальную опасность, так как связаны с памятью уже не принадлежащей к исполняемой программе. Попытка использования такого указателя не приводит к немедленной ошибке. Однако память, на которую он указывает, могла быть уже выделена другому динамическому объекту и попытка записи приведет к порче этого объекта.

```
int *p;  
p= new int;  
*p=55;  
delete p;    // указатель становится "висячим"  
*p=8;        // использование "висячего" указателя
```

Если после **delete p**; сразу написать **p=NULL**;, то в дальнейшем при попытке разыменовать нулевой указатель **p** возникнет исключение, что является более предпочтительным, чем скрытая ошибка изменения участка памяти не принадлежащей программе. Данный прием следует иметь ввиду и после освобождения динамической переменной обнулять указатель:

```
delete p;  
p=NULL;
```

3. "Утечка" памяти.

Данная ошибка возникает, когда память не освобождается, но перестает контролироваться указателем. Подобную ошибку называют "утечкой" памяти, поскольку с такой памятью невозможно ни работать, ни освободить. Такую ошибку трудно обнаружить, поскольку она никак себя не проявляет и не сказывается на работе приложения. Однако при систематических утечках программа требует все больше памяти у операционной системы, замедляя работу других приложений. Далее приводятся две распространенные ситуации, в которых возникает утечка памяти.

Пример. Повторное выделение памяти.

Если выделить память повторно для того же указателя, то ранее выделенная память "утечет" и станет недоступной:

```
int *p;  
p= new int;  
*p=55;  
p= new int;  
//выделяется новый участок памяти под тот же указатель
```

Пример. При выходе из функции указатель p уничтожается, а память с ним связанная не освобождается.

```
void function (int n)  
{ int *p;  
  p= new int;  
  *p=n;  
}
```

Такой фрагмент кода наиболее опасен при использовании локальных переменных в функциях. При неоднократных вызовах функция выделяет новую область памяти, не освобождая ее после использования. В результате может возникнуть ситуация нехватки памяти.

4. Попытка освободить динамическую память, не выделенную ранее

```
int *p;  
delete p;
```

Вызов операции **delete** для неинициализированного указателя игнорируется, не приводя к генерации ошибки.

5. Попытка освободить нединамическую память.

```
int *p,i=55;  
p=&i;  
delete p;
```

При вызове **delete** для нединамической переменной будет сгенерирована ошибка.

Обработка текстов.

Работа со статическими строками

В языке C нет строкового типа данных. C-строка является массивом типа **char**, который заканчивается нулевым символом ("\0").

Определение статической строки, память выделяется при трансляции и в дальнейшем не изменяется :

char имя_массива[n]; где n - количество символов в строке, включая завершающий нуль-символ.

Инициализация строк

Задать значения строкам можно с помощью констант-строк:

```
char stroka [10]="строка 1";  
char stroka [ ]="строка 2";
```

В первом случае под переменную stroka отводится 10 байтов, но используются только первые 7 байтов (включая завершающий нуль-символ "\0").

Во втором случае память выделяется автоматически и ее размер определяется количеством символов строковой константы плюс завершающий нуль-символ. Таким образом переменная stroka занимает 7 байтов.

Массив строк (текст)

Для создания массива строк используется двумерный массив `text[N][M]`:

N – количество строк

M – максимальная длина строки

```
char text [8][7] = {" строка", "и", "текст", "по", "русски", "and", "latin", ""};
```

	0	1	2	3	4	5	6
0	с	т	р	о	к	а	/0
1	и	/0					
2	т	е	к	с	т	/0	
3	п	о	/0				
4	р	у	с	с	к	и	/0
5	а	н	д	/0			
6	l	a	t	i	n	/0	
7	/0						

Пример 1: Ввести текст с клавиатуры (закончить ввод пустой строкой).

Построчная и посимвольная работа с текстом.

```
#include "stdafx.h"
#include <locale>
#include <windows.h>
#include <stdlib.h>
#include <iostream>
#define N 10
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(0, "Russian");
char name[N][80]; // Массив C-строк (N строк по 80 символов)
int i, j, k, count;
cout<<"Введите несколько строк на латинице, окончание ввода - пустая строка"<<endl;
// Построчная работа: при работе со строками используем один(левый) индекс
for (k=0; k<N; k++)
{ gets(name[k]);
// закончить если пустая строка
if (!name[k][0]) // анализ первого символа в строке
break; //name[k][0]==0 (пустая строка)
}
// печать введенного текста
cout<<"С клавиатуры ввели текст : "<<endl;
for (k=0; k<N; k++)
{if (!name[k][0]) break;
else puts(name[k]);
}
cout<<"-----"<<endl;
// поиск одинаковых строк в тексте
for (i=0; i<k; i++) // i определяет «эталон» для поиска
for (j=0; j<k; j++)
if (!strcmp(name[i], name[j]) && i != j)
cout<<"Одинаковые строки в тексте : "<<i<<"="<<j<<endl;
// Посимвольная работа: при работе с отдельными символами используем оба индекса
// подсчет количества цифр в тексте
for (i=0, count=0; i<k; i++) //перебор по строкам
```

```

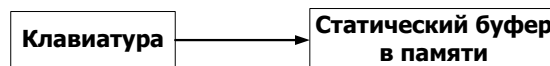
{ //перебор по символам,пока name[i][j]!=0
for (j=0; name[i][j]; j++)
    if (isdigit(name[i][j])) count++;
}
cout<<"Количество цифр тексте = "<<count<<endl;
system("pause");
return 0;
}

```

Динамические строки

Общие принципы работы с динамическими строками

1. Запись данных в буфер



2. Выделение динамической памяти



3. Перенос данных из буфера в динамическую память



Пример 2: Работа с динамической строкой

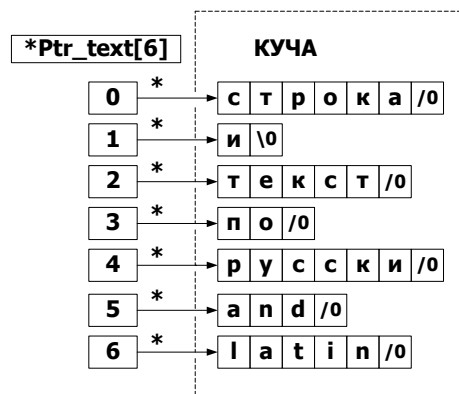
```

#include "stdafx.h"
#include <iostream>
#define N 80
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(0,"Russian");
  char buff[N];           // статический буфер (память выделяется при трансляции)
  char* din_str;          // указатель на строку
  // 1. ввод данных в статический буфер
  printf ("Введите строку на латинице : "); gets(buff);
  // 2. выделение памяти из «кучи»
  din_str = (char*) malloc (strlen(buff)+1);
  printf ("\n Выделенная память не инициализирована: %s \n", din_str);
  // 3. перенос данных в динамическую память
  strcpy (din_str, buff);
  printf ("\n Печать динамической строки : %s \n",din_str);
  system("pause");
  return 0;
}

```


Работа с массивом динамических строк

Массив указателей располагается в статической, а сами строки – в динамической памяти



Пример 3: Выполнить задание **примера 1** с использованием динамических строк и статического массива указателей

```
#include "stdafx.h"
#include <iostream>
#define N 10 // максимальное количество динамических строк
using namespace std;

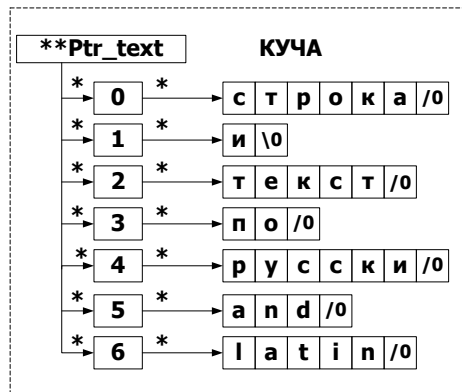
int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(0, "Russian");
    char *ptr_text[N]; // статический массив указателей для адресации динамических строк
    char buff[N]={'\0'}; // буфер для ввода строки
    int i,j,k;
    //динамическую память запрашиваем для каждой строки в цикле
    cout<<"Введите несколько строк на латинице, окончание ввода - пустая строка"<< endl;
    for (k=0; buff[0]!='\0'; k++)
    {
        gets(buff); // ввод строки в буфер
        // выделение динамической памяти под строку
        ptr_text[k] = (char*) malloc (strlen(buff)+1);
        strcpy (ptr_text[k], buff); // копирование строки в динамическую память
    }
    cout<<"-----"<<endl;
    // поиск одинаковых строк в тексте
    // k- количество введенных строк
    for (i=0; i<k; i++)
    for (j=0; j<k; j++)
    if (!strcmp(ptr_text[i],ptr_text[j]) && i!=j)
        cout<<"Одинаковые строки в тексте : "<<i<<"="<<j<<endl;
    char* pstr; //Дополнительный (рабочий) указатель для работы с динамической строкой
    int count;
    for (i=0, count=0; i<k; i++)
    {
        cout<<ptr_text[i]<<endl;
        // установка на i-ую (текущую) строку
        for (pstr = ptr_text[i]; *pstr; pstr++) // инкремент для перемещения в строке
            if (isdigit(*pstr)) count++;
    }
    cout<<"Количество цифр в тексте ="<<count<<endl;
    // освобождение динамической памяти
```

```

for (i=0; i<k; i++)
{ free (ptr_text[i]); ptr_text[i]=0;}
system("pause");
return 0;
}

```

Массив указателей и сами строки – располагаются в динамической памяти



Пример 4: Выполнить задание **примера 1** с использованием динамических строк и динамического массива указателей

```

#include "stdafx.h"
#include <iostream>
#define N 10 // максимальное количество динамических строк
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(0,"Russian");
// Динамический массив указателей на динамические строки
char **dpt; // 2-х уровневый указатель
char buff[80]={'\0'};
int i,j,k, count;
//выделение памяти под динамический массив указателей
dpt= (char**) malloc (sizeof(*dpt)*N);
//выделение памяти и инициализация динамических строк
cout<<"Введите несколько строк на латинице, окончание ввода - пустая строка"<< endl;
for (k=0; buff[0]!='\0'; k++)
{ gets(buff); // ввод с клавиатуры в статический буфер
dpt[k]=(char*) malloc (strlen(buff)+1); //выделение динамической памяти под строку
strcpy (dpt[k], buff); // копирование строки в динамическую память
}
cout<<"-----"<<endl;
// поиск одинаковых строк в тексте, k- количество введенных строк
for (i=0; i<k; i++)
for (j=0; j<k; j++)
if (!strcmp(dpt[i],dpt[j]) && i!=j)
cout<<"Одинаковые строки в тексте : "<<i<<"="<<j<<endl;
char* pstr; // рабочий указатель для работы в динамической строке
for (i=0,count=0; i<k; i++)
{ pstr = dpt[i]; // проверка согласования типов : char* Pstr =(Dpt+i)
for (; *pstr; pstr++)
if (isdigit(*pstr)) count++;
}
}

```

```

}
cout<<"Количество цифр в тексте ="<<count<<endl;
for (i=0; i<k; i++) free (dpt[i]); // возврат памяти динамических строк
free (dpt);                       // возврат памяти массива указателей
system("pause");
return 0;
}

```

Пример 5: Библиотечные функции для посимвольной обработки текста

```

#include "stdafx.h"
#include <iostream>
#include <ctype.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{setlocale(0,"Russian");
    char q[80];
    int k=0, k1=0, k2=0,i;
    cout<<"\nВведите строку на латинице : "; gets(q);
    for (i=0; q[i]!=0; i++) // вычисление количества:
    {    if (isdigit(q[i])) k++;           // цифр
        if (isspace(q[i])) k1++;         // пробельных символов
        if (isupper(q[i])) k2++;         // заглавных букв
    }
    cout<<"Длина строки ="<<i<<endl;
    cout<<"Количество : "<<endl;
    cout<<"\t\tцифр="<<k<<endl;
    cout<<"\t\tпробелов="<<k1<<endl;
    cout<<"\t\tзаглавных букв="<<k2<<endl;
    system ("pause");
    return 0;
}

```

Пример 6: Выполнить различные действия с двумя строками, введенными с клавиатуры.

```

#include "stdafx.h"
#include "stdafx.h"
#include <iostream>
#include <string.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{    setlocale(0,"Russian");
    char s1[80], s2[80];
    cout<<"\nВведите строку на латинице : "; gets(s1);
    cout<<"\nВведите строку на латинице : "; gets(s2);
    // определение длины строк
    cout<<"Длина строки 1: "<< strlen(s1)<<endl;
    cout<<"Длина строки 2: "<< strlen(s2)<<endl;
    // сравнение строк
    //для правильной работы строки должны иметь общие части
    if (strcmp(s1,s2)==0) cout<<"Строки равны"<<endl;
    if (strcmp(s1,s2)>0) cout<<"Строка1 > Строки2"<<endl;
    if (strcmp(s1,s2)<0) cout<<"Строка1 < Строки2"<<endl;
    // объединение строк

```

```

    strcat(s1,s2);
    cout<<"\nОбъединение строк : "<<s1<<endl;
// копирование строк
    strcpy (s1,"Строка-константа");
    cout<<"\nКопирование строки-константы : "<<s1<<endl;
// поиск эталона-символа в строке
    if (strchr(s2,'!'))
    {cout<<"символ ! есть в строке "<<s2;
    cout<<" , он строит на "<<(strchr(s2,'!')-s2+1)<<" месте"<<endl;
    }
        else cout<<"символа ! нет в строке "<<s2<<endl;
// поиск подстроки (эталона-строки) в строке
    char et[]={"ро"}; // эталон для поиска
    if (strstr(s2,et))
    {cout<<"подстрока "<<et<<" есть в строке "<<s2;
    cout<<" , она строит на "<<(strstr(s2,et)-s2+1)<<" месте"<<endl;
    }
        else cout<<"подстроки "<<et<<" нет в строке "<<s2<<endl;
    system ("pause");
    return 0;
}

```

Пример 7: Работа с динамическими матрицами.

```

#include "stdafx.h"
#include <iostream>
#include <iomanip>

// характеристики матрицы
#define n 5 // строки
#define m 4 // столбцы
using namespace std;
// прототип функции «суммирование столбца матрицы»
int sum_stolb_matr (int**, int, int);

int _tmain(int argc, _TCHAR* argv[])
{int **Matr; // указатель на динамическую матрицу
  int i,j;
  // выделение памяти под массив указателей на строки матрицы
  Matr = (int**) malloc(sizeof(*Matr)*n);
  // выделение памяти под строки матрицы
  for (i=0; i<n; i++)
  {Matr[i]= (int*) malloc (sizeof(*Matr[i]));
  // Инициализация и печать
    for (j=0; j<m; j++)
    {Matr[i][j]=rand()%100;
    cout<<setw(10)<<Matr[i][j]<<"\t";
    }
  }
  // вызов функции вычисления суммы столбца (№2)
  cout<<"sum_stolb ("<<2<<")="<<sum_stolb_matr(Matrn,2)<<endl;
  system("pause");
  return 0;
}

```

```
//-----
int sum_stolb_matr (int** matr, int m_m, int k)
{
    int i,s;
    for (i=0,s=0;i<m_m;i++) s+=matr[i][k];
    return s;
}
```

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Функции, работающие с текстами.

Анализ символов – библиотека <ctype.h>

1. Функции проверки отдельных символов (начинаются с is).

Данная группа функций выполняет различные проверки, то есть ставит вопросы, например «Является ли символ буквой?» Результатом является ответ «да» (ненулевое значение) или «нет» (нулевое значение):

int isalpha (int ch) - буква
int islower (int ch) - буква нижнего регистра
int isdigit (int ch) - цифра
int isxdigit (int ch) - шестнадцатичная цифра
int iscntrl (int ch) - управляющий символ (например, \n)
int isspace (int ch) - пробельные символы

2. Перевод букв из верхнего регистра в нижний и наоборот.

int tolower(int c) нижний регистр
int toupper(int c) верхний регистр

Функции возвращают преобразованную букву.

3. Работа со строками - <string.h>

Различают две группы функций:

- Функции, начинающимися с **str**, работают с C-строками (\0 - конец строки).
- Функции, начинающиеся с **mem**, работают с массивами символов, позволяя работать и с нулевыми байтами.

Общее правило:

функции, модифицирующие один из аргументов (напр.: копирование, слияние) всегда изменяют первый аргумент.

1. Функции группы **str** C-строки (\0 - конец строки).

char *strcpy(char *s1, char *s2)
 копирует строку s2 в s1. Возвращает s1.

char *strcat(char *s1, char *s2)
 объединяет строки s1 и s2 (дописывает s2 в s1). Возвращает s1.

int strcmp(char *s1, char *s2)
 сравнивает строки. Возвращает 0 для совпадающих строк, отрицательное значение при s1<s2 и положительное при s1>s2.

int strlen(char *s1)
 Возвращает длину строки s1

char *strchr(char *s1, char sim)
 Возвращает указатель на первое вхождение символа **sim** в строку **s1**

char *strstr(char *s1, char *s2)
 Возвращает указатель на первое вхождение строки **s2** в строку **s1**

Аналогичные функции с контролем длины строки:

char *strncpy(char *s1, char *s2, size_t n)

копирует строку s2 в s1, но копируется не более n символов. Возвращает s1.

char *strncat(char *s1, char *s2, size_t n)

int strncmp(char *s1, char *s2, size_t n)

char *strerror(size_t n)

возвращает строку сообщения, соответствующего ошибке с номером n.

2. Функции группы *mem* аргументы -массивы символов, позволяют работать и с нулевыми байтами.

void *memcpy(void *dst, void *src, size_t len)

копирует len байтов (включая нулевые) из src в dst. Возвращает dst.

void *memmove(void *dst, void *src, size_t len)

делает то же, что и memcpy. Это - единственная функция, которая правильно копирует перекрывающиеся объекты.

int memcmp(void *s1, void *s2, size_t len)

аналог strcmp, но с учетом нулевых байтов.

void *memset(void *s, int c, size_t len)

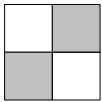

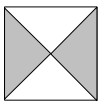
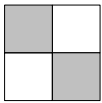
заполняет первые len байтов массива s символом c.

Контрольные вопросы

1. Для чего используется динамическая память в программировании?
2. Как долго хранятся данные в динамической памяти?
3. Какие возможны варианты доступа к динамической памяти?
4. Что возвращает операция выделения динамической памяти в случае успешного выполнения?
5. Что возвращает операция выделения динамической памяти, если участок требуемого размера не может быть выделен?
6. Почему при завершении работы с динамической памятью ее необходимо освободить? Какие могут быть последствия для работы программы, если не освобождать динамическую память?
7. Приведите пример статического выделения памяти под переменную.
8. Приведите пример динамического выделения памяти под переменную.
9. В чем основное различие статического и динамического массива.
10. Какой оператор выделяет динамическую память, приведите пример.
11. Какой оператор освобождает динамическую память, приведите пример.
12. Что представляют собой строки?
13. Каким образом строки описываются и определяются?
14. Какие функции используются для ввода строки и чем они отличаются друг от друга?
15. Какие функции используются для вывода строки?
16. Каким образом можно выделить слово из строки?
17. Где находится описание прототипов функции обработки строк ?

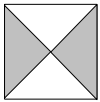
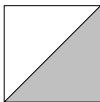
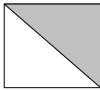
Общие требования к выполнению заданий


1. При оформлении ввода-вывода данных информация на экране должна быть отформатирована:
 - на экран выводится тема задания (кратко);
 - ввод данных и результат вычислений выводить с комментариями;
 - выделять области ввода и вывода информации с помощью строк-разделителей.
2. Данные размещаются в динамической памяти.
3. Обязательное использование пользовательских функций, работающих с массивами и матрицами произвольных размеров.

Варианты задания	Условия задания	
1, 16	<p>Создать функцию для определения самого длинного и самого короткого слова в строке (вернуть 2 указателя)</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово end.</p> <p>Данные размещать в динамической области. Определить строку с самым коротким и с самым длинным словом. Отформатировать текст – выполнить выравнивание по ширине (выравнивать одновременно по левому и правому краю, увеличивая расстояние между словами)</p>	
		<p>Создать динамическую вещественную матрицу NxM (N и M ввести с клавиатуры). Размещение в памяти как в примере 3.</p> <p>Создать функцию, которая возвращает 2 значения : максимум в верхней заштрихованной области и минимум в нижней.</p>
2, 17	<p>Создать функцию для определения слов с заданной длиной, вернуть массив указателей.</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «stop».</p> <p>Данные размещать в динамической области.</p> <p>Найти все слова с заданной длиной (число ввести с клавиатуры).</p> <p>Отформатировать текст – задать длину строки (ввести с клавиатуры) и вывести текст на экран в измененном виде.</p>	
		<p>Создать динамическую вещественную матрицу NxM (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение NxM элементов). Создать функцию, которая возвращает 2 значения : максимум в заштрихованной области и минимум в нижней.</p>
3, 18	<p>Создать функцию для определения количества слов в строке, вернуть массив указателей на начало слов.</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «finish». Данные размещать в динамической области.</p> <p>В каждой строке оставить не более n слов (n ввести с клавиатуры). Убирать наиболее длинные слова</p>	
		<p>Создать динамическую вещественную матрицу NxM (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение NxM элементов). Создать функцию, которая возвращает массив указателей на отрицательные элементы заштрихованной области.</p>
4, 19	<p>Создать функцию для анализа строки - определение количества слов, цифр, пробельных и управляющих символов (сформировать массив результатов)</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «and». Данные размещать в динамической области.</p> <p>Изменить текст: Переставить строки, так чтобы они располагались по увеличению количества слов, в конце каждой строки отразить её характеристики (информацию, которую возвращает функция.)</p>	
		<p>Создать динамическую вещественную матрицу NxM (N и M ввести с клавиатуры). Размещение в памяти как в примере 4.</p> <p>Создать функцию, которая формирует массив результатов - возвращает суммы столбцов заштрихованной области .</p>

5, 20	<p>Создать функцию для определения слов с длиной больше заданного значения (вернуть указатели на найденные слова.)</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «ок».</p> <p>Данные размещать в динамической области. Удалить каждое второе слово с длиной больше заданного значения.</p> <p>Вывести текст на экран и отформатировать:</p> <p>Разбить текст на страницы, параметры (длину строки и количество строк) ввести с клавиатуры.</p> <div data-bbox="316 456 416 591"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти как в примере 3.</p> <p>Создать функцию, которая формирует массив результатов – возвращает указатели на максимальные элементы в заштрихованной области.</p>
6, 21	<p>Создать функцию для определения одинаковых слов в строке (возвращаются указатели на начало слов).</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «по».</p> <p>Данные размещать в динамической области. Одинаковые слова в строках написать заглавными буквами. Вывести текст на экран и отформатировать:</p> <p>Реализовать функцию «выровнять строки по центру».</p> <div data-bbox="316 927 416 1039"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение $N \times M$ элементов). Создать функцию, которая формирует массив результатов – возвращает указатели на нулевые элементы заштрихованной области.</p>
7, 22	<p>Создать функцию для определения в строке слов, начинающихся со строчной буквы (возвращаются указатели на начало слов).</p> <p>Тестовая программа: Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «yes». Данные размещать в динамической области. Внести изменения в текст: слова, начинающиеся со строчной буквы пронумеровать (нумерацию с единицы начинать в каждой строке).</p> <div data-bbox="316 1375 416 1464"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти как в примере 3.</p> <p>Создать функцию, которая формирует массив результатов – возвращает указатели на минимальные элементы заштрихованной области.</p>
8, 23	<p>Создать функцию для анализа строки - определение длины строки, количества слов, строчных и прописных символов (сформировать массив результатов)</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «and». Данные размещать в динамической области.</p> <p>Изменить текст: Переставить строки, так чтобы они располагались по алфавиту, в конце каждой строки отразить её характеристики (информацию, которую возвращает функция.)</p> <div data-bbox="316 1845 416 1957"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти как в примере 3.</p> <p>Создать функцию, которая формирует массив результатов – возвращает указатели на положительные элементы заштрихованной области.</p>

<p>9, 24</p>	<p>Создать функцию для поиска цифр в строке (возвращается массив указателей). Тестовая программа: Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «ok». Данные размещать в динамической области. Изменить текст: четные цифры удвоить, а нечетные удалить из текста. Вывести текст на экран и отформатировать: Разбить текст на страницы, параметры (длину строки и количество строк) ввести с клавиатуры.</p> <div data-bbox="316 421 416 517"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение $N \times M$ элементов). Создать функцию, которая формирует массив результатов – возвращает указатели на минимальные элементы заштрихованной области .</p>
<p>10, 25</p>	<p>Создать функцию для поиска слов в строке (возвращается массив указателей). Тестовая программа: Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «yes». Данные размещать в динамической области. Изменить текст: в четных строках перед каждым словом поставить знак «+», в нечетных строках каждое слово удвоить. Вывести текст на экран и отформатировать: Каждая последующая строка должна содержать на одно слово больше: в первой строке вывести одно слово, во второй – два и т.д.</p> <div data-bbox="316 972 416 1068"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти как в примере 4. Создать функцию, которая возвращает 2 значения : максимум в правой заштрихованной области и минимум в левой.</p>
<p>11, 26</p>	<p>Создать функцию для анализа строки - определение длины строки, количества слов, цифр, пробельных символов (сформировать массив результатов) Тестовая программа: Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «and». Данные размещать в динамической области. Изменить текст: Переставить строки, так чтобы они располагались по наличию цифр (строки с цифрами – в начале текста, без цифр – в конце), в конце каждой строки отразить её характеристики (информацию, которую возвращает функция.) Вывести текст на экран и отформатировать: Разбить текст на страницы, параметры (длину строки и количество строк) ввести с клавиатуры</p> <div data-bbox="316 1554 416 1650"> </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение $N \times M$ элементов). Создать функцию, которая формирует массив результатов – возвращает указатели на отрицательные элементы заштрихованной области .</p>

12, 27	<p>Создать функцию для поиска слов в строке, начинающихся с заданной буквы (возвращается массив указателей на начало слов).</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «end». Данные размещать в динамической области.</p> <p>Изменить текст: в четных строках удалить слова, начинающиеся с заданной буквы, в нечетных строках такие слова написать заглавными буквами</p> <p>Вывести текст на экран и отформатировать:</p> <p>Реализовать функцию «выровнять строки по правому краю».</p>
	<div data-bbox="316 488 416 589">  </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение $N \times M$ элементов). Создать функцию, которая формирует массив результатов – возвращает значения максимальных элементов заштрихованной области .</p>
13, 28	<p>Создать функцию для определения количества слов в строке, вернуть массив указателей на начало слов.</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «end». Данные размещать в динамической области.</p> <p>Изменить текст: в четных строках перед каждым словом поставить «!», в нечетных строках удалить каждое второе слово</p> <p>Вывести текст на экран и отформатировать:</p> <p>Реализовать функцию «выровнять строки по левому краю».</p>
	<div data-bbox="316 1077 416 1178">  </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти как в примере 4.</p> <p>Создать функцию, которая формирует массив результатов – возвращает указатели на значения максимальных элементов заштрихованных областей .</p>
14, 29	<p>Создать функцию для поиска цифр в строке (возвращается массив указателей).</p> <p>Тестовая программа: Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «end». Данные размещать в динамической области.</p> <p>Изменить текст: каждую цифру увеличить на ее порядковый номер (первую цифру на 1, вторую – на 2 и т.п.)</p> <p>Вывести текст на экран и отформатировать: Реализовать функцию «выровнять строки по левому краю».</p>
	<div data-bbox="316 1559 416 1648">  </div> <p>Создать динамическую вещественную матрицу $N \times M$ (N и M ввести с клавиатуры). Размещение в памяти: все строки матрицы располагаются в едином массиве (один запрос на выделение $N \times M$ элементов). Создать функцию, которая формирует массив результатов – минимальные значения с строках заштрихованных областях .</p>

15, 30	<p>Создать функцию для поиска слов в строке, начинающихся с заглавной буквы (возвращается массив указателей на начало слов).</p> <p>Тестовая программа:</p> <p>Ввести текст (несколько строк) с клавиатуры, признак окончания – слово «end». Данные размещать в динамической области.</p> <p>Изменить текст: слова, начинающиеся с заглавной буквы переместить в начало строки. Вывести текст на экран и отформатировать: После точки начинать с «красной строки»</p>
	<div data-bbox="317 454 411 546">  </div> <p>Создать динамическую вещественную матрицу NxM (N и M ввести с клавиатуры).</p> <p>Размещение в памяти как в примере 4.</p> <p>Создать функцию, которая формирует массив результатов - возвращает указатели на нулевые элементы в заштрихованной области .</p>