РАБОТА СО СТРОКАМИ.
ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ.

Определение и инициализация строки

Символьная строка – это массив символов, заканчивающийся нулевым символом. *Нулевой символ* – это символ с кодом 0, что записывается следующим образом ‘\0’. Положение нулевого символа определяет фактическую длину строки. Количество элементов в символьной строке на 1 больше, чем изображение строки.

sizeof(“a”) == 2

Определение символьной строки:

char StringID [SIZE];

StringID – это указатель на первый символ в строке.
SIZE -1 – это максимальное количество символов, которые могут быть записаны в строке.
Инициализация символьной строки:
1. char str[] = “black”;
2. char str[] = “ ’b’ ‘l’ ‘a’ ‘c’ ‘k’ ‘\0’ ”;
3. char *str = “black”;

Функции ввода/вывода символьных строк

1. Функция **scanf(“%s”, StringID);**
Единственный случай, когда в функции **scanf** не записывается адрес перед вводимой переменной – это при вводе символьной строки, т.к. имя строки – это и есть адрес.
Для функции **scanf** ввод завершается либо символом пробел, либо символом '\n ', причем сами эти символы в строку не заносятся, а заменяются на символ конец строки. Следовательно, с помощью функции **scanf** можно вводить только слова.

```
char str[80];

scanf("%s", str);

printf("%s\n",str);
```

2. char* gets(char*s) – считывает только строку s из стандартного потока.
Для функции gets ввод завершается символом '\n ', сам символ '\n' в строку не заносится, а заменяется на символ конец строки. Следовательно, с помощью функции gets можно вводить предложения

```
char str[80];
gets(str);
printf("%s\n",str);
```

3. **int puts(const char* s)** - записывает строку в стандартный поток, добавляя в конец строки символ ‘\n’. В случае удачного завершения возвращает значение больше или равное 0 и отрицательное значение (EOF=-1) в случае ошибки.

```
char str[80];

gets(str);

puts(str);

puts(“Hello!!!!”);
```

Передача строк в качестве параметров функций
Строки при передаче в функции могут передаваться как одномерные массивы типа **char** или как указатели типа **char***.

1. ReturnType FunctionID(char [ ]);

2. ReturnType FunctionID (char *);

Библиотечные функции работы со строками

Для работы со строками существуют специальные библиотечные функции, которые содержатся в заголовочном файле *string.h*.

1. *Функция присваивания***:**

char* ***strcpy*** (**char***s1, **const char***s2);

Копирует строку s2 в строку s1, причем размер строки s1 должен быть достаточно большим, чтобы хранить строку s2 и символ конец строки. Возвращает значение строки s1.

Пример:

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

void main(){
      char str1[80]= "bbb ccc ddd", str2[80];

      strcpy(str2,str1);
      puts(str2);

      strcpy(str1,"aaaaa");
      puts(str1);
}
```

2.1. *Функция добавления***:**

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

Добавляет к строке **s1** строку **s2**. Возвращает значение строки s1.

Пример:

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

void main(){
      char str1[80]="bbb ccc ddd",
           str2[80]="123 456 7";

      strcat( strcat(str1,str2), "!" );
      puts(str1);
}
```

2.2. *Функция добавления:*

char* ***strncat*** (**char***s1, **const char***s2, **int** n);

Добавляет не более **n** символов из строки **s2** в строку **s1**. Возвращает значение строки s1.

Пример:

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

void main(){
      char str1[80]="bbb ccc ddd",
           str2[80]="123 456 7";

      strcat( strncat(str1,str2, 3), "!" );
      puts(str1);
}
```

3.1. *Функция сравнения**:***

int ***strcmp*** **(const char*s1,const char*s2);**

Сравнивает посимвольно строки **s1** и **s2**, *причем регистр учитывается.*

*Возвращает***: 0**, если строки одинаковы;

1 если встречается хотя бы один символ в строке **s1**, код которого больше, чем код символа строки **s2;**

-1, в противном случае.

Пример:

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

void main(){
      char str1[100]="Hello", str2[80]="hello";
```

```
      int k;

      k = strcmp(str1,str2);

      if( !k )
            puts("The same.");
      else
            printf("k = %d\n",k);
}
```

3.2 *Функция сравнения*:

int *strncmp* (const char*s1,const char*s2, int n**);**

Сравнивает не более n символов строк, регистр учитывается.

Пример:

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

void main(){
      char str1[100]="Hello", str2[80]="Hello, Dima";
      int k;

      k = strncmp(str1, str2, 5);

      if( !k )
            puts("The same.");
      else
            printf("k = %d\n",k);
}
```

3.3 *Функции сравнения*:

int *strcmpi* (const char*s1,const char*s2);

В данной функции регистр не учитывается.

Пример:

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

void main(){
      char str1[100]="Hello", str2[80]="hello";
      int k;

      k = strcmpi(str1,str2);

      if( !k )
            puts("The same.");
      else
            printf("k = %d\n",k);
}
```

1. *Функция определяющая фактический размер строки*:

int *strlen* (const char*s);

Возвращает количество символов, предшествующих символу конец строки.

Пример:

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

void main(){
      char str[80];
      int len;

      gets(str);
      len = strlen(str);
      printf("%s\tdlina=%d\n", str, len);
}
```

5. *Функция выполняет поиск первого вхождения подстроки s2 в строку s1.*

char* *strstr* (const char*s1,const char*s2);

Если есть токая подстрока в s1, то функция возвращает указатель на элемент из s1, с которой начинается s2. Если же такой подстроки нет, то возвращает NULL.

Пример:

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

void main(){
      char x[100],y[100], *ptr;
      int kol = 0;

      printf("Введите строку: ");
      gets(x);
      printf("Введите слова для поиска в строке: ");
      scanf("%s",y);

      ptr = strstr(x,y);
      while(ptr){
            kol++;
            ptr=strstr(ptr += strlen(y), y);          }

      printf("k = %d\n",kol);
}
```

Массив строк

Определение массива строк

char StringID [Size1][Size2];

Size1– это целая положительная константа**,** т.е. количество указателей, которые содержат адреса строк.
Size2 – размер каждой строки.
StringID – это **адрес адреса,** т.е. указатель на указатель**.**

```
      char string[5][20];
      int i;

      for(i=0;i<5;i++){
            printf("Строка %d\n",i+1);
            gets(string[i]); // ввод массива строк    }

      for(i=0;i<5;i++)
            puts(string[i]); // вывод массива строк
```

Ввести строки и отсортировать их в алфавитном порядке.

```
      char string[5][20], buf[20];
      int i, j, k;

      for(i=0;i<5;i++){
            printf("Строка %d\n",i+1);
            gets(string[i]);
      }

      for( i = 0; i < 4 ;i++){
         for(k = i, j = i+1; j < 5; j++)
            if(strcmp( string[k], string[j] ) > 0)
                  k = j;

         strcpy(buf,string[i]);
         strcpy(string[i],string[k]);
         strcpy(string[k],buf);
      }

      puts("");
```

```
for(i = 0;i < 5; i++)
    puts(string[i]);
```

Динамическое распределение памяти

Функция выделения памяти

Ядром динамического выделения памяти являются функции, объявленные в стандартной библиотеке в заголовочном файле **stdlib.h**

1. void * *malloc* (unsigned size)

1. выделяет область памяти размером *size* байтов;
2. в случае успеха, возвращает указатель на начало выделенного блока памяти;
3. если для выделенной памяти не хватает места, возвращает **NULL.**

2. void * *calloc* (unsigned num ,unsigned size)

1. выделяет область памяти размером **num*size** байтов;
2. в случае успеха, возвращает указатель на начало выделенного блока памяти;
3. выделяемая память автоматически инициализируется нулями;
4. если для выделенной памяти не хватает места, возвращает **NULL.**

Функция переопределения выделенной памяти

void * *realloc* (void *ptr ,unsigned size)

1. изменяет размер *динамически выделенной памяти*, на которую указывает *ptr* на новый размер - *size* байтов. Т.е для того чтобы можно было изменить размер памяти его необходимо обязательно динамически выделить с помощь функции **malloc** или функции **calloc;**
2. если указатель *ptr* не является значением, которое ранее было определено функциями **malloc** или **calloc**, то поведение функции **realloc** не определено;
3. при изменении размера выделяется новая память, а не меняется размер ранее выделенной памяти, следовательно необходимо переадресовывать указатель;
4. если размер *size* больше, чем размер ранее существовавшего блока, то новое, неинициализированное пространство будет выделено в конце блока и предыдущее содержимое пространства сохраняется;
5. если для переопределения памяти не хватает места, то функция возвращает **NULL** и содержимое пространства на которое указывает ptr, остается нетронутым;
6. если значение size = 0, а ptr не ноль, то функция **realloc** действует как функция **free**.

Функция освобождения памяти

void *free* (void *ptr);

Функция *освобождает* область памяти, ранее выделенной при помощи функции **malloc, calloc** или **realloc** на которую указывает ptr;

1. если ptr- **NULL**, то free ничего не выполняет.
2. если ptr не является указателем, проинициализированным ранее одной из функций выделения памяти, то поведение функции не определено;
3. функция **free** не располагает средствами передачи ошибки, возможно возникающей при ее выполнении.

Примеры выделения памяти

Для того чтобы динамически выделить память необходимо:

1. *Определить указатель, на тот тип для которого выделяется память*

Type *ptr;

2. *Выделить память либо с помощью функции malloc либо с помощью calloc*

2.1 *с помощью функции malloc*

ptr = (*Type* *) malloc (n * sizeof (*Type*));

2.2 *с помощью функции calloc*

ptr = (*Type* *) calloc (n , sizeof (*Type*));

3. *Проверка на выделения памяти*

```
if ( !ptr ){
  puts(“ Not enough memory ”);
  return;
}
```

4. *После использования освободить*:
free (ptr);

Пример выделения символьной строки

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(){
  char x[100],  *ptr;
  int n ,i, j;

  printf("Введите строку: ");
  gets(x);

  n = strlen(x);
  ptr= (char *) malloc ( (n+1) * sizeof ( char ));
  if( ! ptr ){
      printf("Нет места!!!\n");
      exit(1);
  }
. . . . . .
. . . . . .

      free( ptr );
}
```

Пример выделения одномерного массива

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(){
      int *arr;
      int n i;

      printf("Enter the size of array: ");
      scanf("%d", &n);
      if(n <= 0) {
         puts("Errors");
         return;
}
      arr = (int*) malloc ( n * sizeof(int));
      if( !arr ) {
          printf("Not enough memory \n");
          exit(1);
      }
. . . . . .
. . . . . .
      free(arr);
}
```

Пример выделения двумерного массива

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main(){
  int**arr;
  int row, col, i, j;

  printf("Enter row: ");
  scanf("%d", &row);
  printf("Enter col: ");
  scanf("%d", &col);
```

```
  if(row <= 0 || col <= 0) {
      puts("Errors");
      return;
  }

  arr = (int**) malloc ( row * sizeof(int*));
  for(i = 0 ; i < row; i++)
     arr[i] = (int*) malloc (col * sizeof(int));
     if( !arr ) {
         printf("Not enough memory \n");
         exit(1);
     }
. . . . . .
. . . . . .
  for(i = 0 ; i < row; i++)
      free(arr[i]);

  free(arr);
}
```