Chapter 5
Strings
Topics

• What is a String?
• Variable Strings
• Input and Output of Strings
• The String Library
What is a String

- In its simplest form, a string is an array of characters, type char.
- Like all arrays, Strings can be both a pointer and an array.
What is a String

- Strings contain letters, symbols, escape characters, and numbers.
- To be displayed correctly as a string, the last character needs to be a ‘\0’ or a NULL.
What is a String

- Strings have been used before.
- Strings are denoted by being surrounded by double quotes.
What is a String

- The text area and format area of a printf statement is a string.

```c
printf("hello\n");
```
Declaring a String

- Strings can be declared three ways.
- As an array
  - char DataString[100];
  - DataString = “hello”;
Declaring a String

• As a pointer

  • char *DataString="This is a String";
Strings and Memory

• Strings are a memory location
• Strings have the same rules as pointers and arrays
Strings and Memory

- If a string does not end with a ‘\0’, it will run on indefinitely until it stops.
Strings and Memory

- If a string does not have enough room to contain the output of an operation, it will cause an error.
Strings and Memory

- If a string writes into unassigned memory space, it can cause an error or a fault.
Input and Output

- Strings are read in with a format string
- "%s" is the format string for a String

```c
char DataString[100];
scanf("%s", DataString);
// & is not needed since DataString is an Array
printf(DataString);
```
String.h

- The string library is accessed through the string.h header file.
- The code for adding the string library is placed at the beginning of your program in the pre-processor statements.

- #include <string.h>
The length of a string can be found using the `strlen` command.
The quick brown fox jumps over the lazy dog.
StringData is 46 characters long.
**strncpy**

- The `strncpy` takes the value of one string, and copies it into another string, overwriting the contents of the string.
- `strncpy` can only work on a string that has the memory area to store the data.
char *StringData="The quick brown fox jumps over the lazy dog.\n";
char StringCopy[100];
int LoopControl;
for (LoopControl=0;LoopControl<100;LoopControl++)
    StringCopy[LoopControl]='\0';

printf("Before Copy[%s]\n",StringCopy);
strcpy(StringCopy,StringData);
printf("After Copy[%s]\n",StringCopy);

Before Copy[]
After Copy[The quick brown fox jumps over the lazy dog.]
The function **strcat** combines the contents of two strings with concatenation, adding the contents of one string to the end of another.

*strcat*, like *strcpy*, requires the target string to have enough memory to store the contents of both strings.

*strcat(string1,string2);*
strcat

char *StringRoot="A B C D=>";
char *StringAdd="<=1 2 3 4";
char StringCat[100];
int LoopControl;
for (LoopControl=0;LoopControl<100;LoopControl++)
    StringCat[LoopControl]='\0';
strcpy(StringCat,StringRoot);
printf("Before Concatenation[%s]\n",StringCat);
strcat(StringCat,StringAdd);
printf("After Concatenation[%s]\n",StringCat);

Before Concatenation[A B C D=>]
After Concatenation[A B C D=><=1 2 3 4]
The function `strcmp` compares two strings by comparing the characters in them.

1. `result = strcmp(string1, string2);`
2. If the two strings are identical, `strcmp` returns 0.
3. If the first different character in the two strings occurs earlier in order, the result is negative.
4. If the first different character in the two strings occurs later in order, the result is positive.
char *First = "A";
char *Second = "J";
char *Third = "T";

printf("\n");
printf("Strcmp Example\n");
printf("A is Before T: % d\n", strcmp(First, Third));
printf("J is the same: % d\n", strcmp(Second, Second));
printf("T is After A: % d\n", strcmp(Third, First));
printf("\n");
printf("\n");
More Input and Output

- `fgets`- gets the string from a source. Its return is the pointer to the string or a NULL if nothing is read.
- `fputs`- puts the string to a source
More Input and Output

- stdin - a source that is the keyboard
- stdout - a source that is the screen
More Input and Output

char DataLine[1024];
fgets(DataLine,1024,stdin);

This is a line of text

Result in DataLine:
   This is a line of text

char DataLine[1024];
scanf("%s",Dataline);

This is a line of text

Result in Dataline:
   This
More Input and Output

char *Msg="Hello, class"

fputs(Msg, stdout)

Screen Shows:

Hello, class

char *Msg="Hello, class"

printf("%s",Msg);

Screen Shows:

Hello, class
Passing A Double Reference

- char **ptr is a double reference.
- Usually used in a parameter pass
- Used when the string is desired to be altered, it is a pointer to a pointer.
- What does this example code do?

Example:
```c
int countem(char **ptr)
{
    int counter=0;
    while(**ptr==' ')
    {
        counter++;
        (*ptr) ++;
    }
    return counter
}

void main()
{
    char * Msg="     Hello";
    result=countem(&Msg);
}
```
strncpy

- The `strncpy` will copy only the first n characters in a string.
- Like `strcpy`, `strncpy` target string must have the memory area to store the result.
- `strncpy` does not automatically put a null terminator in the string.
strncpy

char DataString[100];
char *CopyString="123456";
int LoopControl;
for (LoopControl=0;LoopControl<100;LoopControl++)
    DataString[LoopControl]=' ';

strncpy(DataString,CopyString,4);
printf("[%s]\n",DataString);
DataString[4]='\0';
printf("[%s]\n",DataString);

[1234
   nYA]
[1234]
strncmp

- The `strncmp` function is the same as the `strcmp` function, except it checks only the first `n` characters in the string.
- Function `strncmp` function returns a positive difference if the second string happens later in alphabetical order.
- Function `strncmp` returns a negative difference if the second string happens earlier in alphabetical order.
char *First ="A";
char *Second="J";
char *Third ="T";

printf("n");
printf("%s is Before %s: % d\n",First,Third,strncmp(First,Third ,1));
printf("%s is the same : % d\n",Second,strncmp(Second,Second,1));
printf("%s is After  %S: % d\n",Third,First,strncmp(Third,First ,1));
printf("n");
printf("n");

• A is Before T: -19
• J is the same : 0
• T is After  A: 19
strncmp

char *Fourth="Blue Skies ";
char *Fifth ="Blue Bottle";
char *Sixth ="Blue Jeans";

printf("\n");
printf("%s compares with %s at % 2d characters: % d\n",Fifth,Sixth,5,strncmp(Fifth,Sixth,5));
printf("%s compares with %s at % 2d characters: % d\n",Fifth,Sixth,10,strncmp(Fifth, Sixth ,10));
printf("%s compares with %s at % 2d characters: % d\n",Fourth,Sixth,10,strncmp(Fourth,Sixth ,10));
printf("\n");
printf("\n");

• Blue Bottle compares with Blue Jeans at 5 characters: 0
• Blue Bottle compares with Blue Jeans at 10 characters: -8
• Blue Skies compares with Blue Jeans at 10 characters: 9
The `strchr` function takes a string and a character value, and returns the pointer to the character if it is found in the string.

If the character is not found in the string, it returns NULL.
strchr

char *DataString="This is the target $";
printf("The $ occurs at %p in %s.\n", strchr(DataString,'$'), DataString);
printf("The ? occurs at %p in %s.\n", strchr(DataString,'?'), DataString);

The $ occurs at 00415667 in This is the target $.
The ? occurs at 00000000 in This is the target $.
The `strstr` function is just like the `strchr` function, except it searches for a string instead of a single character.

- `strstr` returns a `NULL` if the string is not found.
char *DataString="This is the target XYZZY";
printf("The XYZ occurs at %p in %s.\n",
    strstr(DataString,"XYZ"),DataString);
printf("The ABC occurs at %p in %s.\n",
    strstr(DataString,"ABC"),DataString);

• The XYZ occurs at 00415683 in This is the target XYZZY.
• The ABC occurs at 00000000 in This is the target XYZZY.
strpbrk

• The `strpbrk` function finds the first occurrence of one character in the second string that occurs in the first string.

• If no character occurs, then the function returns a NULL.
strpbrk

char *DataString="ABCDEF01234";
printf("A digit occurs at %p.\n",strpbrk(DataString,"0123456789"));
printf("A symbol occurs at %p.\n",strpbrk(DataString,"!@#$%^&*()"));

• A digit occurs at 00415672.
• A symbol occurs at 00000000.
The `strspn` function returns the count of the number of the characters in a set that start a string. If the characters in the set do not start a string, `strspn` returns 0.
strspn

```c
char * DataString = "aardvark";
char * TargetSet  = "ardeiou";
char * SecondSet  = "rdvrk";
printf("%d\n", strspn(DataString, TargetSet));
printf("%d\n", strspn(DataString, SecondSet));
```

- 4
- 0
strcspn

- The `strcspn` function returns the length of the characters that begin a string that are not in the target set.
strcspn

```c
char * DataString = "aardvark";
char * TargetSet  = "ardeiou";
char * SecondSet  = "rdvrk";
printf("%d\n", strcspn(DataString, TargetSet));
printf("%d\n", strcspn(DataString, SecondSet));
```

- 0
- 2
The `strtok` function takes a string and a set of characters called delimiters.

For the first time the function is called, it returns the first section of the string, called a token, marked off by the delimiters.

For subsequent calls, by passing a NULL, the `strtok` function returns each subsequent token in sequence.

When no more tokens can be found, it returns a NULL.
strtok

```c
char DataString[]="Alpha;Beta;Gamma";
char *token=NULL;

token=strtok(DataString,";" );
printf("[\%s]",token);
token=strtok(NULL,";" );
printf("[\%s]",token);
token=strtok(NULL,";" );
printf("[\%s]",token);
token=strtok(NULL,";" );
printf("[\%s]",token);
printf("\n");
```

- [Alpha][Beta][Gamma]