“C Programming”

Strings

Characters and strings

- **Characters**
  - Building blocks of programs - Every program is a sequence of grouped characters
  - Character constant – One character surrounded by single quotes
    - **Example:** ‘A’, ‘?’

Characters and strings

- **Strings**
  - **Definition:** **A sequence of characters** are treated as a single unit, called as string.
    - Can include letters, digits and special characters (*, /, $)
  - **String literal** (string constant) - written in double quotes
    - **"Hello"**
  - **Strings are arrays of characters**
    - String - a pointer to first character
    - Value of string - is the address of first character

Distinction Between Characters and Strings

- The representation of a char (e.g., 'A') and a string (e.g., "A") is completely different.
- A string is an array of characters ended with the null character.

Character 'A'

String "A"

Memory Storage for a String

- The string is always ended with a null character '\0'.
- The characters after the null character are ignored.
- **Example:**

```
char str[20] = "Initial value";
```

Strings - Declarations

- **String declarations**
  - Declare as a character array or a variable of type **char ***

    char name[] = "ram";

    char name2[] = {'r', 'a', 'm'}; // not a string

    char name3[]= {'r', 'a', 'm', 0}; // not a string

    char *nameptr = "ram"; // a constant string
  - Remember that strings represented as character arrays end with **'\0' (A NULL Character)**
- **How to read string as input?**
  - Use **scanf** function

    Example: scanf("%s", name);
    - Copies input into **name[]**
    - Do not need **& (because a string is a pointer)**
  - We must give one character of array for **'\0'**

Strings – Reading and writing

```
//=========================
//Read and write a string
//=========================
#include <stdio.h>
int main()
{
        char name[25];
        printf("\nEnter your name: ");
        scanf("%s", name);
        printf("\nMy first Name is: %s\n\n", name);
        return 0;
}
```

Strings

```
#include<stdio.h>
int main ()
{
	char name[6] = {'A', 'C', '\0', 'T', 'S', '\0'};
	printf("message: %s\n", name );
	return 0;
}
```

Strings – Reading and writing

```
#include <stdio.h>
int main()
{
        char str1[5] = "desd";
        char *str2="ram07";
        char str3[5];
        printf("size of str1 = %d\n", sizeof(str1));
        //string size or array size ???
        printf("size of str2 = %d\n", sizeof(str2));
        //string size or pointer size ???
      // str3 = "DESD";    //correct or Error or Warning???
        return 0;
}
```

getc - putc

```
#include<stdio.h>
int main()
{
        char c;
        printf("Enter character: ");
        c=getc(stdin);

        printf("Character entered: ");
        putc(c,stdout);

        printf("\n");
        return(0);
}
```

getchar - putchar

```
#include <stdio.h>
int main ()
{
   char c;
   int r;
   printf("Enter character: ");
   c = getchar();
   printf("Character entered: ");

   r = putchar(c);
   printf("\nAscii: %d\n",r);
   return(0);
}
```

Strings – Intrinsic Functions

a) Character handling functions
b) String Conversion functions
c) Standard Input and Output functions
d) String manipulation functions
e) Comparison functions
f) Search functions
g) Memory functions
h) Other functions

a) Character Handling Library

- **Character handling library**
  - It Includes the functions to perform useful tests and manipulations on characters
  - Each function receives a character or **EOF** as an argument
  - Many times, a character can be **treated as an integer (ASCII)**
  - **EOF is normally have a value '-1',** but some hardwares do not allow to store negative numbers in integers
- Header File: **<ctype.h>**

a) Character Handling Library…

Prototype	Description
`int isdigit( int c )`	Returns **`true`** if **`c`** is a digit and **`false`** otherwise.
`int isalpha( int c )`	Returns **`true`** if **`c`** is a letter and **`false`** otherwise.
`int isalnum( int c )`	Returns **`true`** if **`c`** is a digit or a letter and **`false`** otherwise.
`int isxdigit( int c )`	Returns **`true`** if **`c`** is a hexadecimal digit character and **`false`** otherwise.
`int islower( int c )`	Returns **`true`** if **`c`** is a lowercase letter and **`false`** otherwise.
`int isupper( int c )`	Returns **`true`** if **`c`** is an uppercase letter; **`false`** otherwise.
`int tolower( int c )`	If **`c`** is an uppercase letter, **`tolower`** returns **`c`** as a lowercase letter. Otherwise, **`tolower`** returns the argument unchanged.
`int toupper( int c )`	If **`c`** is a lowercase letter, **`toupper`** returns **`c`** as an uppercase letter. Otherwise, **`toupper`** returns the argument unchanged.
`int isspace( int c )`	Returns **`true`** if **`c`** is a white-space character—newline (**`'\n'`**), space (**`' '`**), form feed (**`'\f'`**), carriage return (**`'\r'`**), horizontal tab (**`'\t'`**), or vertical tab (**`'\v'`**)—and **`false`** otherwise
`int iscntrl( int c )`	Returns **`true`** if **`c`** is a control character and **`false`** otherwise.
`int ispunct( int c )`	Returns **`true`** if **`c`** is a printing character other than a space, a digit, or a letter and **`false`** otherwise.
`int isprint( int c )`	Returns **`true`** value if **`c`** is a printing character including space (**`' '`**) and **`false`** otherwise.
`int isgraph( int c )`	Returns **`true`** if **`c`** is a printing character other than space (**`' '`**) and **`false`** otherwise.

Isdigit- example

```
#include <stdio.h>
#include <ctype.h>
int main() {
        char var1 = 'h';
        char var2 = '2';
        if( isdigit(var1) )
            printf("var1 = %c is a digit\n", var1 );
        else
            printf("var1 = %c is not a digit\n", var1 );
        if( isdigit(var2) )
            printf("var2 = %c is a digit\n", var2 );
        else
            printf("var2 = %c is not a digit\n", var2 );
        return(0);
}
```

toupper- example

```
#include <stdio.h>
#include <ctype.h>
int main()
{
   int i = 0;
   char c;
   char str[] = "cdac Acts";
   while(str[i]) {
            putchar (toupper(str[i]));
        i++;
   }
   printf("\n");
   return(0);
}
```

```c
/* Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
       isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
       isdigit( '#' ) ? "# is a " : "# is not a ", "digit" );

   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalpha:",
       isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
       isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
       isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
       isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter" );
```

```
 printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalnum:",
       isalnum( 'A' ) ? "A is a " : "A is not a ",
       "digit or a letter",
       isalnum( '8' ) ? "8 is a " : "8 is not a ",
       "digit or a letter",
       isalnum( '#' ) ? "# is a " : "# is not a ",
       "digit or a letter" );

 printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit( 'F' ) ? "F is a " : "F is not a ",
       "hexadecimal digit",
       isxdigit( 'J' ) ? "J is a " : "J is not a ",
       "hexadecimal digit",
       isxdigit( '7' ) ? "7 is a " : "7 is not a ",
       "hexadecimal digit",
       isxdigit( '$' ) ? "$ is a " : "$ is not a ",
       "hexadecimal digit",
       isxdigit( 'f' ) ? "f is a " : "f is not a ",
       "hexadecimal digit" );
   return 0;
}
```

```
Output:
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

b) String Conversion Functions

- Conversion functions
  - In **<stdlib.h>** (general utilities library)
- Convert strings of digits to integer and floating-point values

Prototype	Description
`double atof( const char *nPtr )`	Converts the string **nPtr** to **double**.
`int atoi( const char *nPtr )`	Converts the string **nPtr** to **int**.
`long atol( const char *nPtr )`	Converts the string **nPtr** to long **int**.
`double strtod( const char *nPtr, char **endPtr )`	Converts the string **nPtr** to **double**.
`long strtol( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **long**.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **unsigned long**.

```c
/* Using atof function*/

#include <stdio.h>
#include <stdlib.h>
int main()
{
  double d;
  d = atof( "99.0" );
  printf("%s%.3f\n%s%.3f\n", "The string \"99.0\" converted to
              double is ", d, "The converted value divided by 2
              is ", d/2.0 );
  return 0;
 }
```

```
Output:
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

C) Standard I/O Functions

- Used to manipulate characters and strings
- Header file is: **<stdio.h>**

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array **s** until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in **c**.
`int puts( const char *s );`	Prints the string **s** followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to **printf**, except the output is stored in the array **s** instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to **scanf**, except the input is read from the array **s** instead of reading it from the keyboard.

Gets and puts- example

```
#include <stdio.h>
int main()
{
   char str[50];
   printf("Enter a string : ");
   //scanf("%s", str);
   gets(str);
   //    printf("You entered: %s", str);
   //   printf("\n");
   puts(str);
   printf("\n");
   return(0);
}
```

d) String Manipulation Functions

- String handling library has functions to
  - Manipulate string data
  - Search strings
  - Tokenize strings
  - Determine string length

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string **s2** into array **s1**. The value of **s1** is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most **n** characters of string **s2** into array **s1**. The value of **s1** is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most **n** characters of string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.

```c
/* Using strcat and strncat */
#include <stdio.h>
#include <string.h>

int main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
   printf( "strncat(s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
   return 0;
}
```

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

Class room work

- Write a c program to concatenate two strings without using library functions
- Write a c program to copy string without using library functions
- Write a c program to find length of string without using library functions

e) Comparison Functions

- **Comparing strings**
  - Computer compares numeric ASCII codes of characters in string

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

  - Compares string **s1** to **s2**
  - Returns a negative number if **s1 < s2**, zero if **s1 == s2** or a positive number if **s1 > s2**

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

  - Compares up to **n** characters of string **s1** to **s2**
  - Returns values as above

f) Search Functions

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character **c** in string **s**. If **c** is found, a pointer to **c** in **s** is returned. Otherwise, a **NULL** pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string **s1** consisting of characters not contained in string **s2**.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string **s1** consisting only of characters contained in string **s2**.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string **s1** of any character in string **s2**. If a character from string **s2** is found, a pointer to the character in string **s1** is returned. Otherwise, a **NULL** pointer is returned.
`char *strrchr( const char *s, int c );`	Locates the last occurrence of **c** in string **s**. If **c** is found, a pointer to **c** in string **s** is returned. Otherwise, a **NULL** pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string **s1** of string **s2**. If the string is found, a pointer to the string in **s1** is returned. Otherwise, a **NULL** pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to **strtok** breaks string **s1** into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string **s2**. The first call contains **s1** as the first argument, and subsequent calls to continue tokenizing the same string contain **NULL** as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, **NULL** is returned.

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


int main () {
   const char mains[20] = "cdac acts course";
   const char sub[10] = "acts1";
   char *ret_val;

   ret_val = strstr(mains, sub);
        if(ret_val != NULL)
   printf("The substring %s   is present in the main string
%s\n", sub,mains);
        else
   printf("The substring %s  is not present in the main string
%s\n", sub,mains);

   return(0);
}
```

```c
/*Using strerror */

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

int main()
{
   printf( "%s\n", strerror( 2 ) );
   return 0;
}
```

```
No such file or directory
```

Note: The error message related to the error number 2 is "No file or directory".

THANK YOU