CHARACTER PROCESSING AND STRINGS

The Data Type *char*

- Each character is stored in a machine in one byte (8 bits)
  - 1 byte is capable of storing 2^8 or 256 distinct values.
- When a character is stored in a byte, the contents of that byte can be thought of as either a character or as an integer.

The Data Type *char*

- A character constant is written between single quotes.

  ‘a’

  ‘b’
- A declaration for a variable of type char is

  char c;
- Character variables can be initialized

  char c1=‘A’, c2=‘B’, c3=‘*’;

In C, a character is considered to have the integer value corresponding to its ASCII encoding.

lowercase	‘a’	‘b’	‘c’	…	‘z’
ASCII value	97	98	99	…	122
uppercase	‘A’	‘B’	‘C’	…	‘Z’
ASCII value	65	66	67		90
digit	‘0’	‘1’	‘2’	…	‘9’
ASCII value	48	49	50	…	57
other	‘&’	‘*’	‘+’	…	
ASCII value	38	42	43		

Characters and Integers

- There is no relationship between the character '2' (which has the ASCII value 50) and the constant number 2.
- '2' is not 2.
- 'A' to 'Z' 65 to 90
- 'a' to 'z' 97 to 112
- Examples:
  - printf("%c",'a');
  - printf("%c",97); have similar output.
  - Printf("%d",'a');
  - printf("%d",97); have also similar output.

The Data Type *char*

- Some nonprinting and hard-to-print characters require an escape sequence.
- For example, the newline character is written as **\n** and it represents a single ASCII character.

Name of character	Written in C	Integer Value
alert	\a	7
backslash	\\	92
double quote	\"	34
horizontal tab	\t	9

Input and Output of Characters

- **getchar ( )** reads a character from the keyboard.

```
c = getchar();  /* variable c contains the
                   next character of input */
```

- **putchar ( ):** prints a character to the screen.

```
putchar(c);   /* prints the contents of the
                 variable c as a character */
```

```
/* Illustrating the use of getchar( ) and putchar( ) */

#include <stdio.h>
int main (void)
{
   char c;
   while ((c=getchar()) != EOF) {
      putchar(c);
      putchar(c);
   }
}
```

```
abcdef
aabbccddeeff
```

EOF : It is control-d in Unix; control-z in DOS.

```
/* Capitalize lowercase letters and
 * double space */

int main(void)
{  int c;
   while ((c=getchar()) != EOF){
      if ('a' <= c && c <= 'z')
         putchar(c+'A'-'a'); /*convert to
uppercase*/
      else if (c == '\n'){
         putchar ('\n');
         putchar ('\n');
      }
      else putchar (c);
   }
}
```

cop3223!c C

Character Functions

Function	Nonzero (true) is returned if
`isalpha(c)`	c is a letter
`isupper(c)`	c is an uppercase letter
`islower(c)`	c is a lowercase letter
`isdigit(c)`	c is a digit
`isalnum(c)`	c is a letter or digit
`isspace(c)`	c is a white space character

Function	Effect
`toupper(c)`	changes c to uppercase
`tolower(c)`	changes c to lowercase
`toascii(c)`	changes c to ASCII code

```
/* Capitalize lowercase letters and double space */
#include <stdio.h>
#include<ctype.h>

int main(void)
{  int c;
   while ((c=getchar()) != EOF){
      if (islower(c))
         putchar(toupper(c)); /*convert to uppercase */
      else if (c == '\n'){
         putchar ('\n');
         putchar ('\n');
      }
      else  putchar (c);
   }
}
```

STRINGS

Fundamentals of Strings and Characters

- **Characters**
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An **`int`** value represented as a character in single quotes
    - **`'z'`** represents the integer value of **`z`**
- **Strings**
  - Series of characters treated as a single unit
    - Can include letters, digits and special characters (**`*`**, **`/`**, **`$`**)
  - String literal (string constant) - written in double quotes
    - **`"Hello"`**
  - Strings are arrays of characters in C
    - String is a pointer to first character
    - Value of string is the address of first character

Strings

- A string constant such as “a string” is an array of characters.
- Each element of the array stores a character of the string.
- In its internal representation, the array is terminated with the null character ‘\0’ so that the end of the string can be found easily.
- Thus, the length of the array is defined one more than the number of characters between the double quotes.

Declaring Strings

```
char myString[10];

myString[0] = ‘H’;
myString[1] = ‘e’;
myString[2] = ‘l’;
myString[3] = ‘l’;
myString[4] = ‘o’;
myString[5] = ‘\0’;
```

0	1	2	3	4	5	6	7	8	9
‘H’	‘e’	‘l’	‘l’	‘o’	‘\0’	?	?	?	?

Initializing Strings

- Character arrays can be initialized when they are declared :

```
char name[5] ={‘M’,’E’, ‘T’,’U’,’\0’};
char name[5] = “METU”; /*compiler
                automatically adds ‘\0’ */
char name[] = “METU”; /*compiler calculates
                the size of the array */
```

Strings and Pointers

- We can declare and initialize a string as a variable of type **`char *`**

```
char *color = “blue”;
```

- But the interpretation is different. “blue” is stored in memory as a string constant. The variable `color` is assigned the address of the constant string in memory.
- If we declare it as:

```
char c[] = “blue”;
```

the array `c` contains the individual characters followed by the null character.

Inputting Strings

- Using subscripts:

```
char c, name[20];
int i;
for (i = 0; (c = getchar())!=‘\n’; i ++)
 name[i] = c;
name[i]=‘\0’;
```

- Using scanf and %s format:

```
scanf(“%s”, name);
```

  - no need to use & operator
  - it will skip the leading blanks in the input, then characters will be read in. The process stops when a white space or EOF is encountered.
  - Remember to leave room in the array for **`'\0'`**

Printing Strings

- Using %s format:

```
printf(“%s %s\n”, “Nice to meet you”, name);
```

- Using subscripts: e.g. printing `name` backwards

```
for (--i; i>=0; --i)
 putchar(name[i]);
putchar(‘\n’);
```

Examples

- **`printf("***Name:%8s*Lastname:%3s*** \n",“John", “Smith");`**
- **Output:**

```
***Name:    John*Lastname:Smith***
```

- **`printf("***%-10s*** \n", “John");`**
- **Output**

```
***John      ***
```

- **`scanf("%d%s%d%s", &day,month,&year,day_name);`**
- **Example input:**

```
      5          November 2001      Monday
```

String Handling Functions (`string.h`)

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

Function prototype	Function description
`char *strcpy( char *s1, char *s2 )`	Copies string **`s2`** into array **`s1`**. The value of **`s1`** is returned.
`char *strncpy( char *s1, char *s2, int n )`	Copies at most **`n`** characters of string **`s2`** into array **`s1`**. The value of **`s1`** is returned.
`char *strcat( char *s1, 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, char *s2, int 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.

String Handling Functions (cont.)

- **`unsigned strlen(char *s);`**
  - A count of the number of characters before \0 is returned.
- **`int strcmp(char *s1, 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(char *s1, char *s2, int n );`**
  - Compares up to **`n`** characters of string **`s1`** to **`s2`**
  - Returns values as above

`strcpy()` and `strncpy()`

- We cannot change the contents of a string by an assignment statement.

```
char str[10];
str = “test”;  /*Error! Attempt to change the base
                 address*/
```

- Thus, we need to use string copy functions
  - `strcpy(str, “test”); /*contents of str changed*/`
  - `strncpy(str, “testing”, 5);`
    `str[5] =‘\0’;  /* str contains “testi” only */`
  - `strcpy(str, “a very long string"); /*overflow of array boundary */`

`strcat()` and `strncat()`

```
char s[8]=“abcd”;
strcat(s,”FGH”);   // s keeps abcdFGH

char t[10]=“abcdef”;
strcat(t,”GHIJKLM”);  //exceeds string length!

strncat(t, "GHIJKLM",3);
t[9] = '\0';      // t keeps abcdefGHI
```

strcmp() and strncmp()

- We can compare characters with <,>,<= etc.

  e.g. 'A' < 'B'
- But we cannot compare strings with the relational operators.

  e.g. `str1 < str2` will compare the memory addresses pointed by `str1` and `str2`.
- Therefore we need to use string comparison functions.

```
strcmp("abcd", "abcde") ->returns a negative number
strcmp("xyz", "xyz")    -> returns zero
strcmp("xyz", "abc")    -> positive number
strncmp("abcde", "abcDEF", 3) -> zero
strncmp("abcde", "abcDEF", 4) -> positive number
```

Examples

```
char s1[] = "beautiful big sky country";
char s2[] = "how now brown cow";
```

Expression	Value
strlen(s1)	25
strlen(s2+8)	9

Statements	What is printed
printf("%s", s1+10);	big sky country
strcpy(s1+10, s2+8)	
strcat(s1, "s!");	
printf("%s",s1);	beautiful brown cows!

```
#include <stdio.h>
#include <string.h>
#define LENGTH 20
/* A string is a palindrome if it reads the same backwards and
  forwards. e.g. abba, mum, radar. This programs checks whether a
  given string is palindrome or not.

int isPalindrome(char s[]);  // function prototype
int main()
{
    char str[LENGTH];

    // read the string
    printf("Enter a string ");
    scanf("%s", str);

    // Check if it is a palindrome.
    if (isPalindrome(str))
       printf("%s is a palindrome.\n", str);
    else
       printf("%s is not a palindrome.\n", str);
 }
```

```
int isPalindrome(char str[])
{
    int i, j, flag;

    i = 0;                  // index of the first character
    j = strlen(str) - 1;    // index of the last character
    flag = 1;               //assume it is a palindrome
    while ((i<j) && flag){
                        // compare the ith and jth. characters
       if (str[i] != str[j])
          flag = 0;  // if not same then string cannot be a
                     //palindrome.
       else {
          i++;
          j--;
       }       // advance to next characters
    }
    return flag;
}
```

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

#define LENGTH 20

 // This program converts a positive integer to a binary
 // number which is represented as a string. For instance
 // decimal number 12 is 1100 in binary system.

void toBinary(int decVal, char *);  //function prototype
int main()
{
    int num;
    char bin[LENGTH];

    // read a positive integer
    printf("Enter a number: ");
    scanf("%d",&num);

    // Convert the number and print it.
    toBinary(num, bin);
    printf("Binary equivalent of %d is : %s",num,
            bin);
}
```

```
void toBinary(int decVal, char *sb) {

    char s0[LENGTH], s1[LENGTH];

    // create an empty string.
    strcpy(sb,"");
    if (decVal == 0)
       strcat(sb,"0");  // if number is zero result is 0
    else                // otherwise convert it to binary
       while (decVal != 0) {
          strcpy(s0,"0");
          strcpy(s1,"1");
          if (decVal%2 == 0)
             strcpy(sb,strcat(s0,sb)); //last character is 0
          else
             strcpy(sb,strcat(s1,sb));  //last character is 1
          decVal = decVal / 2;  /* advance to find the next digit */
       }
    return sb;
}
```