Ch-8 Strings

A string is a 1-D character array terminated by a null ('\0')
↳ This is null character

null character is used to denote string termination
characters are stored in contiguous memory locations.

Initializing strings

Since string is an array of characters, it can be initialized as follows:

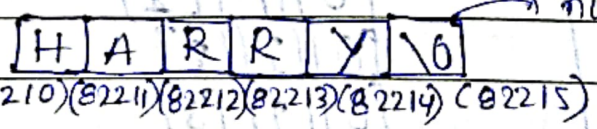
```
char S[] = {'H', 'A', 'R', 'R', 'Y', '\0'};
```

There is another shortcut for initializing strings.

```
char S[] = "HARRY";
```
⇒ In this case C adds a null character automatically

Strings in ~~Memory~~ Memory

A string is stored just like an array in the memory as shown below



Contiguous blocks in memory

Quiz: Create a string using " " & print its content using a loop.

Code:
```
#include <stdio.h>
int main() {
    char str[] = "Harry";
    char *ptr = str;
    while (*ptr != '\0') {
        printf("%c", *ptr);
        ptr++;
    }
    return 0;
}
```

Output
Harry

Code:
```
#include <stdio.h>
int main() {
    char *ptr = "Harry";
    printf("%s", ptr);
    return 0;
}
```

Output
Harry

strncpy: Is source string is longer than target string, the overflow characters are discarded automatically. strncpy(target, source, target_len-1);
target[target_len-1] = '\0';

Printing Strings
A string can be printed character by character using printf & %c.
But there is another convenient way to print strings in C:

```
char st[] = "Harry";
printf("%s", st); ⇒ prints the entire string
```

Taking string input from user
We can use %s with scanf to take string input from the user

```
char st[50];
scanf("%s", st);
```

scanf automatically adds the null character when the enter key is pressed.

Code:
```
#include <stdio.h>

int main() {
    char s[34];
    printf("Enter your name: ");
    scanf("%s", s);
    printf("Your name is %s", s);
    return 0;
}
```

Output
Enta your name
Advait.

Your name is Adwait

Note:
1) The string should be short enough to fit into the array.

2) scanf cannot be used to input multi-word strings with spaces.

gets() & puts()
gets() is a function which can be used to receive a multi-word string.

char st[30];
gets(st); ⇒ The entered string is stored in st!

* Multiple gets() calls will be needed for multiple strings

Likewise, puts can be used to output a string

puts(st); ⇒ prints the string places the cursor on the next line

Declaring a string using pointers
We can declare strings using pointers

char *ptr = "Harry";

char*ptr & char ptr[] are same.

This tells the compiler to store the string in memory & assigned address is stored in a char pointer

Note:

1) Once a string is defined using char st[] = "Harry", it cannot be initialized to something else.

2) A string defined using pointers can be re-initialized.

ptr = "Rohan"

Code:

```
#include<stdio.h>
int main(){
    char ptr[] = "Harry";
    ptr = "Shubham";
    printf("%s", ptr);
    return 0;
}
```

This is not okay, but if you write char* ptr = "Harry" at place of char ptr[] = "Harry" it gives output as Shubham.

Standard Library functions for strings:

string cannot be copied using library functions. If source string is longer than target, the overflow characters may occupy the memory space used by other variables.

C provides a set of standard library functions for string manipulation.

Some of most commonly used string functions are: <string.h>

strlen()

This function is used to count the number of characters in the string excluding the null ('\0') character

```
int length = strlen(st);
```

These functions are declared under <string.h> header file

strcpy()

This function is used to copy the content of second string into first string passed to it.

```
char source[] = "Harry";
char target[30];
strcpy(target, source); ⇒ target now contains "Harry"
```

Target string should have enough capacity to store the source string

strcat()

This function is used to concatenate two strings

BAD: The value of these other variables would seem change spontaneously. On rare occassions, such overflow would generate a run time error messag

char S1 [11] = "Hello"
char S2 [ ] = "Harry"

strcat (S1, S2); ⇒ S1 now contains "HelloHarry" (No space in bet")

Strcmp ()

This function is used to compare 2 strings. It returns: 0 if strings are equal. Negative value if first strings mismatching characters. ASCII value is not greater than second string's corresponding mismatching character. It returns positive values otherwise.

Strcmp ("For", "Joke"); → Positive value

Strcmp ("Joke", "For"); → Negative value

Code:

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

int main() {
char st1 [45] = "Hello";
char *st2 = "Harry";
int val = strcmp (st1, st2);
printf ("Now the val is %d", val);
return 0;

}
```

Output

Now the val is 1

Insight: Here the ascii values of e & a are subtracted & thereafter if it's a +ve value we get 1

Ch-8 → Practice set

Q1) Write a program to take strings as an input from the user using %c & %s. Confirm that the strings are equal.

Code:

```
#include <stdio.h>

int main(){
char st1[34];
char st2[34];
char c;
int i=0;

printf("Enter the value of first string \n");
scanf("%s", st1);
printf("Enter the value of second string character by character \n");

while(c != '\n'){
  fflush(studen);
  scanf("%c", &c);
  st2[i] = c;
  i++;
}
st2[i] = '\0';
```

printf("The value of st1 is %s \n", st1);
printf("The value of st2 is %s \n", st2);

return 0;

}

Output:
Enter
Enter the value of first string
this

Enter the value of second string
character by character
t
h
i
s

The value of st1 is this
The value of st2 is this.

Q2) Write your own version of strlen function from <string.h>

#include<stdio.h>

int strlen(char* st){
    char *ptr = st;
    int len = 0;

```
while (*ptr != '\0') {
    len++;
    ptr++;
}

return len;

}

int main() {
    char st[] = "Harry";
    int l = strlen(st);
    printf("The length of the string is
                                %d", l);
    return 0;
}
```

Output
The length of the string is 5

Q3) Write a function slice() to slice a string. It should change the original string such that it is now the sliced string. Take m & n as the start & ending position for slice.

```
#include <stdio.h>

void slice(char *st, int m, int n) {
    int i = 0;
    while((m+i)<n) {
        st[i] = st[i+m];
        i++;
    }
```

```
    st[i]='\0';
}

int main(){
char st[]="Harry";
slice(st,1,4);
printf("%s",st);
return 0;
}
```

Output
arr

Q4) Write a program to encrypt streng by adding 1 to the ascii value of its characters

```
Code: #include<stdio.h>

void encrypt(char *c){
char *ptr=c;
while(*ptr!='\0'){
  *ptr=*ptr+1;
  ptr++;
}
}

int main(){
    char c[]="Harry";
    encrypt(c);
    printf("Encrypted streng is: %s",c);
    return 0;
}
```

Output:
Encrypted string is : Ibssz

15) Write a program to decrypt the string encrypted using encrypt function in problem 6.

```
#include <stdio.h>

void decrypt(char *c){
    char *ptr = c;
    while(*ptr != '\0'){
        *ptr = *ptr - 1;
        ptr++;
    }
}

int main(){
    char c[] = "Ibssz";
    decrypt(c);
    printf("Decrypted string is: %s", c);

    return 0;
}
```

Output:
Decrypted string is: Harry

16) Write a program to count the occurence of a given character in a string

```
#include <stdio.h>
int occurence(char st[], char c){
    char *ptr = st;
    int count = 0;
    while(*ptr != '\0'){
        if(*ptr == c){
            count++;
        }
        ptr++;
    }
    return count;
}
int main(){
    char st[] = "Hassy";
    int count = occurence(st, 'H');
    printf("Occurences = %d", count);
    return 0;
}
```

Output:

Occurences = 1