Tutorial Sheet 8   
ESC101 – Fundamentals of Computing

**Revision (ask for doubts)**

1. **String:** notion of substring, empty substring, EOF non-character (ASCII value -1), string.h functions (strlen, strcpy, strcat, strstr, strchr, strncpy, strncat). *WARNING*: some functions like strupr, strlwr unavailable in Clang.
2. **Number systems**: notion of place-value system. Octal (%o), decimal (%d), hexadecimal (%x or %X), binary. Ability to write the same integer value in different number systems.
3. **Memory storage**: memory organized in bits and bytes. 8 bits = 1 byte. Using the sizeof operator to get sizes of various datatypes (char 1B, int/float 4B, long/double 8B, pointers 8B).
4. **Pointers**: notion of internal addresses of variables, referencing variables using the & operator, dereferencing pointers using the \* operator and use in expressions/printf, use of pointers in scanf (pass pointers directly) and printf (print address as a long or else value at location after dereferencing).

**Pointers and arrays**

float a[10];

if(a == &a[0])

printf("Same address");

When we declare an array (of any type: float, char etc), the array name is itself a pointer to the first location of the array.

Addresses are always non-negative integers and pointers store these internally as 8 byte integers. This means we can actually do some cool arithmetic with addresses (some operations don’t make sense).

**Assignment and Comparison with Pointers**

char c;

char \*ptr, \*qtr;

ptr = &c; // Both ptr and

qtr = ptr; // qtr point to c

We can compare two pointers using == (the addresses will get compared). We can assign the address stored in one pointer to another pointer simply using the = operator.

**Addition and Subtraction with Pointers**

We can also perform addition and subtraction with pointers but the + and - operators do not work as usual with pointers.

1. char ptr = 000023; // Just for example. Never hardcode addresses

ptr++; // ptr now stores 000024

ptr += 2; // ptr now stores 000026

ptr--; // ptr now stores 000025

1. int qtr = 000133; // Just for example. Never hardcode addresses

qtr++; // qtr now stores 000137

qtr += 2; // qtr now stores 000145

qtr--; // qtr now stores 000141

1. double rtr = 001143; // Just for example. Never hardcode addresses

rtr++; // rtr now stores 001151

rtr += 2; // rtr now stores 001167

rtr--; // rtr now stores 001159

**RULE OF THUMB**

If **ptr** is a pointer to a variable **var** and stores an address **add** then

**ptr += k** will change the address to **add** **+ k\*sizeof(var)**

**ptr –= k** will change the addresss to **add** **– k\*sizeof(var)**

This seemingly funny behavior is actually worth gold and diamonds when used in array and string manipulations ☺

**WARNING:** multiplication \*, division /, and remainder % are considered **illegal operations with pointers**! Only =, ==, +, -, +=, -= are valid.

**Sample Questions to discuss**

**Warning:** string.h has functions strchr which returns a pointer to the first occurrence of a character in a string. However, if that character not present at all, NULL returned as a way of saying that character is not present. Be careful. Same with strstr which searches for a substring within a string.

**Take a string and print it from the fourth character onward**

char str[] = "Hello World";

char\* ptr = str; // str points to str[0]

ptr += 3; // Move it 3 char forward

printf("%s", ptr); //lo World

**Take a string and print it from the point after the first space occurs**

char str[] = "HelloWorld";

char\* ptr;

ptr = strchr(str, ' ');

if(ptr != NULL){ //Space present

ptr++; // Don’t print the space

printf("%s", ptr);

}

Be careful. If space not present at all in the string, then do not print anything at all.

**Take a string print all indices of space character in the string**

char str[] = "Hello ESC 101 !";

char\* ptr = str;

while(strlen(ptr) > 0){

ptr = strchr(ptr, ' ');

if(ptr == NULL) //No more spaces

break;

printf("%d ", ptr - str);

ptr++; // Move on

}

Nice! String functions like printf, strlen, strchr etc, when given pointer to a character in the middle of the string, just start processing from thereon ☺

**Some Pitfalls and recognizing compiler error messages**

1. Keep character arrays sufficiently large to be able to absorb user input, as well as the delimiting NULL character.
2. Do not write risky code confusing %c and %s: for example printf("%s", 'x'); or printf("%c", "abcdef");
3. Do not dereference a pointer without first assigning to a valid address – may cause segfaults. Risky program examples below

long \*p = NULL;

printf("%ld", \*p);

char \*s;

printf("%s",s);

int j = 42;

int \*ptr = j;

int \*ptr;

\*ptr = 100;

1. Never hardcode addresses in your program. Remember, Mr C reserves several addresses for himself and the operating system (including the NULL address). Your variables will keep getting assigned different addresses when you run your programs again and again.

int \*ptr = 10342425;

\*ptr = 100;//Segfault

int j;  
ptr = &j;

\*ptr = 100;//Correct