Course: 2014

playlist:https://www.youtube.com/watch?v=2NWeucMKrLI&list=PL6gx4Cwl9DGAKIXv8Yr6nhGJ9Vlcjyymq

Tutorial 1: Compiler : code we write -> 10101010 (code that computer understands)

Tutorial 2:
Naming convention: Project -> capital CamelCase.
Build and run button: converts program to binary(machine language) then runs it.

Tutorial 3 & 4:
Computer program is made of small pieces called functions:

code	Explanation
`#include <stdio.h>` `#include <stdlib.h>`	Include files to get basic default functions(for e.g: printf()) Technically called: “Preprocessor directives”
`int main()`	main() function is the start point of any program
`{`	Function starts here
`printf("Hello world!\n");`	Print text on the screen, “ \n” = go to a new line, “ \t” = tab , “ \a” = alert (make a sound) \n,\t,\a, etc are called Escape sequences
`return 0;`	Stop the function here
`}`	Function ends here

N.B: Each line inside the function is a piece of instruction and the semicolon ”;” is what indicates the end of this piece of instruction

Tutorial 5:
Commenting : For a paragraph : /* */ For a line //

Tutorial 6: (Conversion Characters :Placehoders)
String = bunch of characters
Conversion character is a placeholder that you can use to put sth. in your string.

Examples:

Code	**Explanation(Output)**
Strings:	
`printf("%s is the best person ever\n","Bucky");`	Removes %s and put the string "Bucky" instead so the output will be : `Bucky is the best person ever`
`printf("%s is the best %s ever\n","Bucky", "programmer");`	`Bucky is the best programmer ever`
Numbers: 1)Integers:	
`printf("I ate %d corndogs last night",9);`	`I ate 9 corndogs last night`
2)float:	
`printf("Pi is %f\n",3.14159265359);`	`Pi is 3.141593`
`printf("Pi is %.2f\n",3.14159265359);`	`Pi is 3.14` (shows no. till 2 decimal places with approx.)
`printf("Pi is %.4f\n",3.14159265359);`	`Pi is 3.1416`

Tutorial 7: Variable: is a placeholder for sth. else. Name convention: small letter/word
Don't name a variable any basic function (such as: main, printf)

Examples:

`int age;`
 	
`age = 27; //age=2014-1987;`
 	
`printf("Bucky is %d years old", age);`
	- Make a varaible called age of data type integar
	- Assign this variable to 27 or 2014-1987
	`Bucky is 27 years old`

`int age;`
`int currentyear;`
`int birthyear;`
`currentyear=2014;`
	A small program that calculates current age of person

`birthyear= 1987;` `age = currentyear-birthyear;`	

Tutorial 8 & 9:

- When you calculate string length or how much memory you need for a string you need a string terminator.
- 13 characters = 14 bytes in memory (extra byte for string terminator)
- When creating/declaring a string we create an array of characters

Examples

`char name[14] = "Bucky Roberts";` `printf("My name is %s\n", name);`	- Declaring a string with size of 14 bytes `My name is Bucky Roberts`
`name[2]= 's';` `printf("My name is %s\n", name);`	- Change the 3rd element in the array `My name is Busky Roberts`

`char food[]="tuna";`
 	
`printf("My food is %s\n", food);`
	-Declaring a string without mentioning the size (not necessary to mention)
	`My food is tuna`

`strcpy(food,"bacon");`
 	
`printf("My food is %s\n", food);`
	- Assign a new string(`"bacon"`) to the `food` array
	`My food is bacon`

Tutorial 10:

Before the program compiles , it runs the preprocessor directives, in other words the code will take whatever inside the header files (`<stdio.h>` and `<stdlib.h>`) and put it & run it before the code to be compiled. This can be done by `include` process directive. Another process directive `#define` is used with constants (convention: UPPERCASED).
N.B: Semicolon is not needed after any processor directive

Example:

`#include <stdio.h>` `#include <stdlib.h>`	copy the code inside these files and substitute it here (before the program(compiling)) <> means to search the default place where headers are
`#define MYNAME "Tuna McButter"`	Make a constant MYNAME and assign it to Tuna McButter
`printf("My name is %s",MYNAME);`	Replace MYNAME by Tuna McButter

before running the code.
Output: `My name is Tuna McButter`

Create a header file: File>New>Empty File
Name Convention: CapitalCamel case
We made a new header file called `BuckysInfo.h` which contains this code:

```
#define MYNAME "Bucky"
#define AGE 28
```

then in the main file:

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

```
#include "BuckysInfo.h"
```

The `""` means: search in the same directory or folder which is a little bit faster than `<>`.

```
int main() {
```

```
int girlsAge = (AGE/ 2) + 7;
```

`girlsAge` will be assigned to the `AGE(28)` in `BuckysInfo.h` divide by 2 and add 7

```
printf("%s can date girls %d or
older",MYNAME,girlsAge);}
```

`Bucky can date girls 21 or older`

Tutorial 11
In order to read data we must use `scanf()`.

Examples:

```
char name[20];
scanf("%s", name);
```

Read a string from user and assign the value of it to the variable `name`

```
int number;
scanf("%d", &number);
```

Read an integer from user and assign the value of it to the variable `number`

`scanf` pauses your program and waits until you press enter.
We need to include this symbol ”`&`” (ampersand) before any variable except array(array has built-in ampersand) (related to pointers)

Tutorials 12 - 15

Some math & coding hints:

- % - modulus(remainder)
- int divided by int = int. float divided by float = float.
- Whatever is between Parenthesis () is done first.
  For example: 4 + 2 *6 = 16 but (4+2)*6 = 48
- `number *=1.1;` is equivalent to `number =number *1.1;`

We can assign 3 integers to the same value as follows:

```
int a;
int b;
int c;
a=b=c=100;
```

Simple program to calculate the average of 3 persons:

```
float age1,age2,age3,average;
age1=age2=4.0;
age3=6;
average= (age1+age2+age3)/3;
```

Tutorial 16 - Typecasting:

```
int num1= 10;
int num2= 3;
float quotient= num1 / num2;
printf("%f ",quotient);
```

Since we are dividing 2 integers.

output:
`3.000000`

```
int num1= 10;
int num2= 3;
float quotient= (float)num1 / num2;
printf("%f ",quotient);
```

Since num1 is type casted into float.

output:
`3.333333`

Tutorials 17 - 22

[If conditions]

```
if (condition1){
//run code here if condition1 is true
}
if (condition2){
//run code here if condition2 is true
}
```

if conditions we can use:

if(num1 > num2)	Checks if num1 greater than num2	if(num1 >= num2)	Checks if num1 greater than or equal num2

if(num1 < num2)	Checks if num1 smaller than num2	if(num1 <= num2)	Checks if num1 smaller than or equal num2
if(num1== num2)	Checks if num1 equals to num2	if(num1!= num2)	Checks if num1 not equal to num2
if(condition1 && condition2)	Checks if both condition1 & condition2 are true	if(condition1 || condition2)	Checks if either condition1 or condition2 is true (or both)

Nested if conditions:

```
if (condition1){
      if (condition2){
      //run code here if condition1 and condition2 are true
      }
}
```

Hint: Always put space between " and %

If and else conditions:

```
if (condition1){
//run code here if condition1 is true
}
else {
//run code here if condition1 is false
}
```

if-else conditions	Example: Check if grades average is A,B,C,D or F

`if(condition1){`
`  //run code here if condition1 is true`
`} else if (condition2){`
`  //run code here if condition1 is false and condition2 is true`
`} else if (condition3){`
`  //run code here if conditions 1,2 are false and condition is true`
`} else if (condition4){`
`  //run code here if conditions 1,2,3 are false and condition is true`
`}else {`
`  //run code here if conditions 1,2,3,4 are false (default)`
`}`
	`float grade1,grade2,grade3;`
	`grade1=90;`
	`grade2=50;`
	`grade3 =67;`
	`float avg = (grade1 + grade2 + grade3)/3;`
	`if(avg >= 90){`
	`  printf("Grade: A");`
	`} else if (avg >= 80){`
	`  printf("Grade: B");`
	`} else if (avg >= 70){`
	`  printf("Grade: C");`
	`} else if (avg >= 60){`
	`  printf("Grade: D");`
	`}else {`
	`  printf("Grade: F");`
	`}`

Tutorial 23

You can test if a character is smaller than another example:
`if('A'<'B')` → true `if('D'>'B')` → true and so on
N.B: there is a difference between lowercase and uppercase letters.

`(condition) ? run code here if condition is true: run code here if condition is false;`

Example 1:

```
(num1>num2) ? printf("num1 > num2"): printf("num1 <= num2");
```

Example 2:

```
int friends=1;
printf("I have %d friend%s",friends,(friends!=1) ? "s":"");
```

If friends =0 the output will be : `I have 0 friends`
If friends =1 the output will be : `I have 1 friend`

Tutorial 24

Increment operator(++) example:

`int a=5, b=10,answer=0;` `answer= ++a * b;`	Since `++a` is compiled, the a will be incremented first, then `answer` will be calculated, therefore answer = 6 * 10 = 60

`printf("Answer %d \n", answer);`	`Answer 60`
`a =5, b=10,answer=0;` `answer= a++ * b;`	Since `a++` is compiled, `answer` will be computed first, then `a` will be incremented, therefore answer = 5 * 10 = 50
`printf("Answer %d \n", answer);`	`Answer 50`

Tutorials 25 - 27 loops

While loop:

```
while(condition){
run code here while condition is true (or until condition is
false)
}
Example:
```

`int tuna = 1 ;`
`while(tuna<5){`
`  printf("tuna is now %d \n",tuna);`
`  tuna++;`
`  }`
	`tuna is now 1`
	`tuna is now 2`
	`tuna is now 3`
	`tuna is now 4`
	`tuna is now 5`

Do- while loop:

```
do{
run the code here for the first time without checking the
condition, then from the second time the code here runs while
condition is true (or until condition is false)
}while(condition);
```

For loop:

we implement for loop to implement a certain code a certain number of times.

```
for(statement(mostly assignment); condition; statement(mostly
increment)){
//your code here
}
```

Example:

`for(int i=0; i < 5; i++){`
`  printf("%d\n",i);`
`}`
	`0`
	`1`
	`2`

This code will run `printf("%d\n",i)` 5 times.	3 4

Tutorial 28

How to create a table:

```
for(int rows=1; rows<=6; rows++){
```

The outer for loop (in red) will repeat the inner for loop(in blue) six times printing a line after the inner loop is finished each time.

```
for(int columns=1; columns<=4;
columns++){

      printf("%d ",columns);
    }

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

The inner loop will print the variable columns (denoting each column) and increment it four times, on one line

Output:

```
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

Tutorials 29 - 30

`break` keyword stops(discontinues) any loop at a certain condition

Example:

`for(int i=0; i < 5; i++){`
`  printf("%d\n",i);`
`  if(i==3){`
`  break;`
`  }`
`}`
	Output will stop at 3:
	0
	1
	2
	3

`continue` keyword skips/ignores any code segments after it in a loop.

Example:

`for(int i=0; i < 5; i++){`
`  if(i==3){`
`  continue ;`
`  }`
`  /*this part will be`
`  ignored when i=3*/`
`  printf("%d\n",i);`
`}`
	Output will skip 3:
	0
	1
	2
	4

Tutorial 31

Switch is alternative to if or if-else statement but it is done to check on one variable.
For example:

`Char grade = 'C';`
`switch(grade){`
`  case 'A' : printf("Excellent \n");`
`break;`
`  case 'B' : printf("very good \n");`
`break;`
`  case 'C' : printf("good \n");`
`break;`
`  case 'D' : printf("Sufficient\n");`
`break;`
`  case 'F' : printf("Failure \n");`
`break;`
`  default : printf("Not`
`available");}`
	This code prints a string corresponding to a certain grade.
	After `switch` we put the variable that we check for (which is `grade`).
	`case 'A' : printf("Excellent \n");` corresponds to
	`if(grade == 'A') {`
	`printf("Excellent \n"); }`
	`break` is put to ensure that the compiler won't continue checking the rest of the cases if it met any.
	`default is optional but if it exists it will run if no previous condition is met.`

Tutorial 32 - 35

`isalpha(char character)` → checks if a certain character is alphabet
`isdigit(char character)` → checks if a certain character is a digit
`isupper(char character)` → checks if a certain character is a uppercase
`toupper(char character)` → converts a character to uppercase if it is lowercase ,otherwise it will leave it the same.
`strcat(char[] string1, char[] string2)` → concatenates/add string2 to string1

For Example :	`char string1[100]="Take ";`
	`char string2[100]="Care";`
	`strcat(string1,string2);`
	`printf("%s \n",string1);`
		The output will be:
		`Take Care`

N.B: make sure that string1 is big enough to hold the strings to be concatenated.

`strcpy(char[] string1, char[] string2)` → replaces the content of string1 with the content of string2 (as if: string1 = string2 , however one array can't be assigned to another)
We must include the <string.h> header file to use strcat and strcpy functions.

`gets(char[] string)` → scan input and assign it to string. Better than scanf because it can deal with String separated by space (for example: "Bill Gates").
`puts("Any string")` → prints a string with a newline.

Tutorial 36-37

Some functions in the <math.h> library:

`ceil(float number)` → rounds up a float/double number
`floor(float number)` → rounds down a float/double number
`abs(int number)` → gets the absolute of a number (removes -ve sign)
`pow(int base, int exponent)` → get the base powered by the exponent. (for example: pow(X,Y) gets X power Y as: X^Y)
`sqrt(int number)` → gets the square root of a number. (sqrt(int 25) → $\sqrt{25}$)
`rand()` → generates a random number.
to generate a random number with a certain range you can use (`rand()% rangemax)+1`
For example: to get a number between 1 & 6 → (`rand()% 6)+1` , as `rand()%6` will give a number between 0 and 5 , then by adding 1 the number will be between 1 & 6.

Tutorials 39-40

Tutorial 41 - [Sorting Algorithms]

Bubble sorting : the following code keeps swapping elements of the array until it is sorted

```
while(1){
```
Infinite loop

```
 swapped = 0;
```
A variable to determine if at least one element in the array is swapped.

```
for(i=0; i< howMany-1;i++){
```
Loop until the element before the end of the array

```
    if(array[i] > array[i+1]){
```
If the current element bigger than the next element

```
    int temp= array[i];
        array[i]=array[i+1];
        array[i+1] = temp;
```
Swap these elements with each other (keep swapping until we reach the last element)

```
            swapped =1 ;
            }
        }
```

Set swapped to true (indicating that at least one element is swapped)

```
if(swapped ==0){
            break;
                }
           }
```

If no element is swapped (the array is sorted) Break the infinite loop

Tutorial 42 - pointers

The address of any variable can be accessed by using the symbol & (ampersand). For e.g:

`printf("%p ", &variable);`	`000000000061FE14` (memory address)

In order to declare a pointer:

`int * pVariable = &variable;`	The asterisks * indicates that pVariable is a pointer and the memory location of the variable is assigned to it.

Name convention of a pointer is p attached to the variable's name in capital case.

Tutorial 43 - dereference pointer

if we try to print *pVariable , we will get the value inside the variable that pVariable stores its memory address (this is called dereferencing a pointer). For example:

`int variable = 19;`	
`int *pVariable = &variable;`	Store the memory address of variable in the pointer pVariable.
`printf("%d ", *pVariable);`	The output will be: `19`

we can also change the value of variable by changing that of *pVariable.

`*pVariable = 71;`	
`printf("\n *pVariable: %d", *pVariable);`	`*pVariable: 71`
`printf("\n variable: %d", variable);`	`variable: 71`

Summary:

`int * pVariable = &variable;` This line will do the following:

&pVariable → %p → address of pointer
pVariable → %p → address of variable

*pVariable → %d → access the value of variable

dereferencing:
*pVariable → pVariable address → variable address → value of variable

Tutorial 44

The array name is a pointer for the first element in the array. So by default dereferencing the array name will give the value of the first element. Also dereferencing the array name plus one will get the value of the second element and so on. For example:

`int i;`
`int array[5]= {7,9,43,21,3};`
 	
`printf("Element \t Address \t \t Value \n");`
 	
`for(i=0;i<5;i++){`
`  printf("array[%d] \t %p \t %d \n",i, &array[i], array[i]);`
`  }`
	Initialize an array and print each element address corresponding to its value. Output:
	Element Address Value
	array[0] 000000000061FE00 7
	array[1] 000000000061FE04 9
	array[2] 000000000061FE08 43
	array[3] 000000000061FE0C 21
	array[4] 000000000061FE10 3

`printf("\n array \t\t %p \n",array);`	**Print the address of the array which is the address of first element:** array 000000000061FE00
`printf("\n *array \t\t %d \n",*array);`	**Derefrenece first element:** *array 7
`printf("\n *(array+2) \t\t %d \n",*(array+2));`	**Derefrenece first element + 2 (3rd):** *(array+2) 43

Tutorial 45 - Strings and Pointers

The array name is considered to be a constant pointer, therefore after initializing an array we can't change its value by directly assigning it to another value because it is as if saying that we want to change the address of the array neglecting elements inside it so elements stored at a certain address will be lost (or couldn't be referred to by any address). For example:

if	`char string1[]="string1";`
We can't compile these lines:	`string1 = "string2";` `string1[] = "string2";`
We can either use	`strcpy(string1,"string2");`

or we can change each element	`string1[0] = "s"`

However we can do the following:

`char * string1="string1";`	Here string1 became a variable pointer so we can change it. And string1 only stores the address where "string1" begins.
`puts(string1);`	However if we tried to print string1 it starts from the address of the first character and print until reaching the null zero
`char string1 = "string2";`	So now we can freely change this pointer variable to any other string.

Tutorial 46

`fgets(char *string, int length, FILE * stream)` → another way of taking input(instead of scanf()) but with limiting the number of characters entered (so that the program won't crash) and with specifying the stream .
For example:
`fgets(pString, 20 ,stdin);` → this will allow the user to only enter 20 characters via standard input (keyboard).

Tutorial 47 - heap

Heap is a leftover/extra memory that we can borrow whenever we need it and give it back whenever the program ends.

In order to borrow the memory we use: malloc(how much memory do we need?) → allocate memory or get memory from the heap. For example to store 5 integers in the heap we use:
`int * points= (int *) malloc (5 * sizeof(int));`

Go to the heap reserve place for 5 integers type cast into pointer int then assign it to the int pointer points. N.B: `sizeof()` gets the size of a certain data type as it may vary across different operating systems or computers.

After we finish using this memory space , we must give it back to the computer using this function: `free(points)`

Example using heap - let the user enters the size of an array and the elements of that array, then calculate the average:

```
int i , howMany, sum;
float average = 0.0;
int * pointsArray;
```

Initializing variables:

`i` → counter, `howMany` → size of array , `sum` → sum of elements in the array, `pointsArray` → integer pointer for the array

```
printf("How many numbers you want to
average?\n");
scanf(" %d",&howMany);
```

Print a string, then read from the user the number of the elements of the array and store it in `howMany` .

```
pointsArray=
(int *) malloc(howMany * sizeof(int));
```

Reserve a place for `howMany` integers in the heap(memory) and make `pointsArray` store the first element address of that array

```
printf("Enter the numbers: \n");
for(i=0; i<howMany; i++){
        scanf(" %d", &pointsArray[i]);
        sum+= pointsArray[i];
}
```

Keep scanning all the elements of the array while getting their sum `howMany` times.

```
average = (float) sum/ (float) howMany;
printf("Average is %f", average);
```

Get the average and print it.

So what is new here that we created a kind of dynamic array as its size is determined by the user.

Tutorial 49 - Structures

We can use structures to group some variables/attributes under one structure.
For example, We can define a user to have these attributes: id, first name , last name , age and weight. Mainly structures are initialized in header files ,so we can define a structure called `user` using the word `struct` in a header file. After defining a struct we can create a new user (struct) and access its members(attributes) in the main file. Check the following table for the implementation:

In header file	In main file	

`struct user{`
`  int id;`
`  char firstName[25];`
`  char lastName[25];`
`  int age;`
`  float weight;`
`};`
	`#include "header.h"`
	`int main(){`
	`  struct user user1;`
	`  user1.id=1;`
	`  user1.firstName = "Elon";`
	`  user1.lastName = "Musk";`
	`  user1.age = 64;`
	`  user1.weight = 70.0;`
	`  return 0; }`
		Create a new user struct `user1`
		set user1 members: `id`,`firstName`,`lastName`, `age` and `weight`

Tutorial 50 - 53: Files

Files could be accessed in 2 types of ways:
Sequential access file → data created in order.
Random access file → store data all over the place(doesn't need to be in a specific order).
When dealing with a file we always need a file pointer (`FILE * fPointer`) to keep track of where we are in the file.
File functions:

1) `FILE * fopen( const char * filename, const char * mode )` → this function open/create a certain file in the same directory of the main file and do sth. to it depending on the mode (`"w"` → write, `"r"` → read, `"a"` → append).
2) `int fprintf(FILE *fp,const char *format)` → this function prints a text inside a file. It takes the pointer of the file and a string.
3) `int fclose( FILE *fp )`→closes a file and frees memory back to the computer.
4) `feof(FILE *fp )` → is used to find the end of a file.

Write a file	Read a file	Append a file

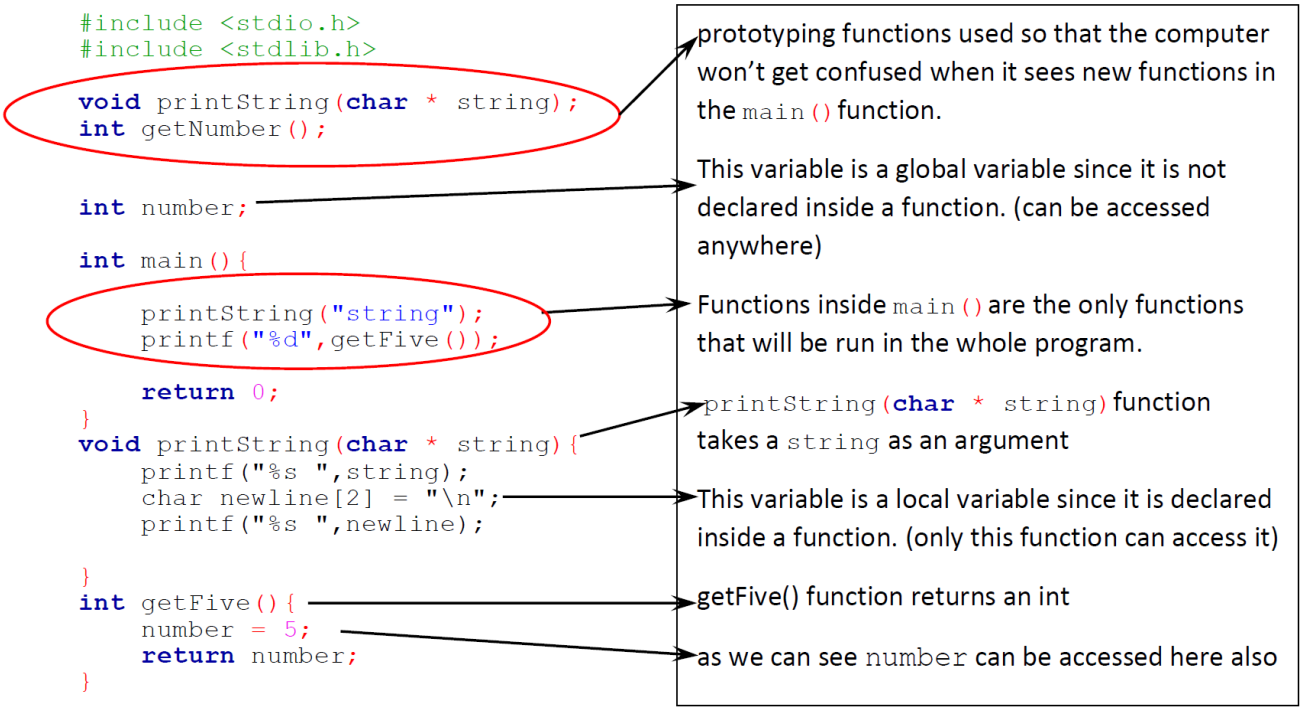
`FILE * fPointer;`
`fPointer =`
`fopen("file.txt", "w");`
 	
`fprintf(fPointer,"This is a new file");`
 	
`fclose(fPointer);`
	`FILE * fPointer;`
	`fPointer =`
	`fopen("file.txt", "r");`
	
	`char singleLine[150];`
	
	`while(!feof(fPointer)){`
	
	`fgets(singleLine,150,fPointer);`
	`puts(singleLine);`
	`}`
	`fclose(fPointer);`
		`FILE * fPointer;`
		`fPointer =`
		`fopen("file.txt", "a");`
		
		`fprintf(fPointer, "\nAdded text");`
		
		`fclose(fPointer);`

N.B: in this mode(`"w"`) if we use `fprintf()` more than once it will replace the text each time it is used.	`while(!feof(fPointer))`→ keep looping until the end of the file `fgets(singleLine,150,fPointer)`→ read a line from the file. - The code prints each read line from the file until we reach its end.	N.B: in this mode(`"a"`) if we use `fprintf()` more than once it will add to the text each time it is used.

5) `int fseek(FILE *stream, long int offset, int whence)` →go to a certain place in the file . stream is the pointer of ther file , offset is the place you want start from (+ve or -ve), whence is from where you want to start(`SEEK_SET` → beginning of file, `SEEK_CUR` → Current position of the file pointer, `SEEK_END` → end of file)
   For example:
   `fseek(fPointer,7,SEEK_SET)`→ goes to 7th place in a text from the beginning
   `fseek(fPointer,-7,SEEK_END)`→ goes to 7th place in a text from the end.

Tutorial 54 - 58

The following code demonstrates functions, global & local variables, arguments and return:



The output of the code above will simply be:

```
string
5
```

Pass by value	Pass by reference
`void passByValue(int i){ i=99; return; }`	`void passByAddress(int *i){ *i=99; return; }`
In the upper function the value of i is passed as an argument for the function , which means that i remains as it is.	In the upper function the address of i is passed as an argument for the function , which means that i will change its value since it is being dereferenced.
`int x = 20; passByValue(&x); printf("Passing by value, x is %d\n",x);`	`int x = 20; passByAddress(&x); printf("Passing by address, x is %d\n",x);`
Output will be: `Passing by value, x is 20`	Output will be: `Passing by address, x is 99`