Create a symbols and integrate multiple symbols

Organised & Supported by **RuggedBOARD**

Agenda

- Functions
- Library Functions
- User-defined functions
- Function call by value & call by reference
- Local & Global Variables
- Storage Classes
- Command line Arguments

Functions

Def:
A self contained block of code performing a specific job of some kind.
Meaning:
'Do something' in a program.
Repesentation:
Any name followed by a pair parenthesis.
Syn:
Functionname ( )

Two types of functions:
Library functionctions:
Pre-defined functions. are called'Library functions'.These are defined in the header files.
User-defined functionctions:
These functions are defined by programmer only.

Library Functions

1	**printf()** This function is used to print the all char, int, float, string etc., values onto the output screen.
2	**scanf()** This function is used to read data from keyboard.
3	**getc()** It reads character from file.
4	**gets()** It reads line from keyboard.
5	**getchar()** It reads character from keyboard.
6	**puts()** It writes line to o/p screen.
7	**putchar()** It writes a character to screen.

8	**fopen()** All file handling functions are defined in stdio.h header file.
9	**fclose()** Closes an opened file.
10	**getw()** Reads an integer from file.
11	**putw()** Writes an integer to file.
12	**fgetc()** Reads a character from file.
13	**putc()** Writes a character to file.
14	**fputc()** Writes a character to file.

Library Functions

```
#include <math.h>
#include <stdio.h>
// Driver code
int main()
{
        double number, squareRoot;
        number = 12.5;
        // Computing the square root
        squareRoot = sqrt(number);
        printf("Square root of %.2lf = %.2lf",number, squareRoot);
        return 0;
}
```

Note: Compile above program with –lm option

```
gcc  -c file.c
gcc  file.o -o file -lm
```

User-defined Functions

main() is a function.
It has collection of function.
Each and every C program must start its execution from this function only.

- Any number of functions can be used in a program.
- A function can be used any number of times in a program.
- Any function can return a value with return statement.

Here, **return** is a key word.
Syn: return (var / constant / exprn) ;
- It can be used as the last statement in every definition.
- The void function does not return any value.
- The word void is also a keyword.
- A function can be called from another one.

```
main()
{
// codes start from here
}
```

Arguments Types: Actual and Formal

Actual arguments :

The vars., which are passing through a function called 'Actual arguments'

Syn:

functionname (argslist) ;

Eg. add(a, b);

Here, the variables 'a' and 'b' are called actual args.

Formal arguments:

The variables, which are declared in a pair of parentheses at the function definition called 'Formal arguments'.

Syn:

```
returntype  functionname(args list)
{
// body of the function.
}
```

Eg:

```
void add ( int x, int y)
{
// body
}
```

Here, the vars. 'x' and 'y' are called formal arguments.

Rules for calling a specific function

Rules for calling a specific function:

1.Calling function name should be matched with function name in the function definition.

2.No. of Actual arguments = No. of Formal arguments.

3.The order, data types and returns must be matched.

User-defined function

Types of User-defined functions :

There are 4 types of user-defined functions

- without passing args., with no return
- without passing args, with return
- with passing args. , with no return
- with passing args., with return

```
#include<stdio.h>
void sum(void);
int main ()
{
   sum ();
   return 0;
}
void sum ()
{
   int a,b,c;
   printf("enter 2 numbers:\n");
   scanf ("%d%d", &a, &b);
   c = a+b;
   printf("sum = %d\n",c);
}
```

```
#include<stdio.h>
int sum (void);
int main ()
{
   int c;
   c= sum ();
   printf("sum = %d\n",c);
   return 0;
}
int sum ()
{
   int a,b,c;
   printf("enter 2 numbers:");
   scanf ("%d%d", &a, &b);
   c = a+b;
   return c;
}
```

```
#include<stdio.h>
void sum (int, int );
int main ()
{
   int a,b;
   printf("enter 2 numbers :");
   scanf("%d%d", &a,&b);
   sum (a,b);
   return 0;
}
void sum ( int a, int b)
{
   int c;
   c= a+b;
   printf ("sum=%d\n", c);
}
```

```
#include<stdio.h>
int sum ( int,int);
int main ()
{
   int a,b,c;
   printf("enter 2 numbers: ");
   scanf("%d%d", &a,&b);
   c= sum (a,b);
   printf ("sum=%d\n", c);
   return 0;
}
int sum ( int a, int b )
{
   int c;
   c= a+b;
   return c;
}
```

Function call by value & call by reference

-The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function.

-In this case, changes made to the parameter inside the function have no effect on the argument.

C programming language uses **call by value** method to pass arguments. In general, this means that code within a function cannot alter the arguments used to call the function.

- Call by reference method of passing arguments to a function copies the address of an argument into the formal parameter.

- Inside the function, the address is used to access the actual argument used in the call.

-This means that changes made to the parameter affect the passed argument.

- To pass the value by reference, argument pointers are passed to the functions just like any other value.

Local & Global Variables

- Variables that are declared inside a function or block are called local variables.
- Used only by statements that are inside that function or block of code.
- Local variables are not known to functions outside their own.

```
int d;
int main ()
{
     /* local variable declaration */
      int a, b;
      int c;
}   //Here all the variables a, b and c are local to main() function.
```

- Global variables are defined outside of a function, usually on top of the program.
- The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.
- A global variable is available for use throughout your entire program after its declaration.

```
int d;
int main ()
{
     /* local variable declaration */
      int a, b;
      int c;
}   //Here variable d global  to main() function.
```

Storage Classes

Storage class refers to the permanence or longevity of a variable and its scope within the program, that is the part of the program over which the variable is recognized.

C supports the following storage classes –

- automatic (local)
- extern (global)
- static
- register

The general syntax for declaring a variable is

storage-class data-type variable_name

Automatic

- Automatic variables are always declared within a function and are local to the function in which they are defined.

- Any variable declared within a function is interpreted as an automatic variable unless a different storage-class specification is included within the declaration.

-Automatic variables defined in different functions are independent of one another, even if they have the same name.

```
#include <stdio.h>
int main()
{
   auto int a = 28;
   int b = 8;
   printf("The value of auto variable : %d\n", a);
   printf("The sun of auto variable & integer variable : %d\n", (a+b));
   return 0;
}
```

Extern

The **extern** storage class does not create a variable, but merely informs the compiler of its existence. Their scope extends from the point of definition through the remainder of the program.

When a global **extern** declaration is made (outside a function), it indicates that the variable referred to is declared in another file, which will be linked with the file containing the **extern** declaration.

For example, the global declaration
extern int ram;

Format:
extern data_type variable_name;

Within this program file, **ram** is a global variable.
The actual storage for **ram** is allocated in another program file and in that file ram is also global.

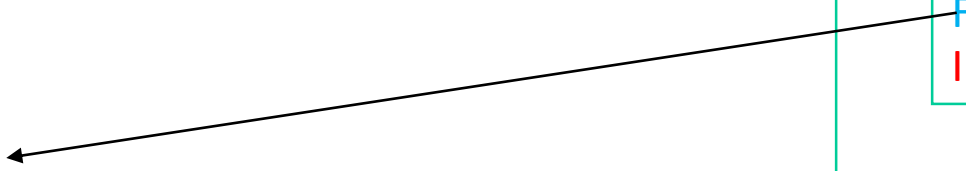
Extern

Filename : extern.c

```
#include <stdio.h>
#include"extern.h"
extern int x;
int b = 8;
int main()
{
   auto int a = 28;
   x = 32;
   extern int b;
   printf("The value of auto variable : %d\n", a);
   printf("The value of extern variables x and b : %d,%d\n",x,b);
   x = 15;
   printf("The value of modified extern variable x : %d\n",x);
   return 0;
}
```

Filename:extern.h

```
Int x;
```



Static Variables

- Static variables are local to a particular function but their values are retained across function calls.
- They remain in existence throughout the life of the program rather than coming and going each time the function is activated.

```
#include <stdio.h>
void func(void);
int main()
{
  func();
  func();
  func();
  return 0;
}

void func(void)
{
      int num1=1;
      static int num2;
      num1++;
      num2++;
      printf("num1 is : %d \n static num2 is : %d\n",num1,num2);
}
```

Register

The compiler can be told to keep a variable in one of the machine's registers, instead of the memory, Since a **register** access is much faster than a memory access, keeping the frequently accessed variables like loop control variables in the register, will lead to faster execution of programs.

```
#include <stdio.h>
int main()
{
   register char x = 'S';
   register int a = 10;
   auto int b = 8;
   printf("The value of register variable b : %c\n",x);
   printf("The sum of auto and register variable : %d\n",(a+b));
   return 0;
}
```

Command line arguments

- It is possible to pass some values from the command line to your C programs when they are executed.
- These values are called **command line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.
- The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   int arg;
   for(arg = 0; arg < argc; arg++) {
      if(argv[arg][0] == '-')
         printf("option: %s\n", argv[arg]+1);
      else
         printf("argument %d: %s\n", arg, argv[arg]);
   }
   return 0;
}
Execution:./cmd_line -i -lr 'hi there' -f file.c
```

Thank You