~~Array in c Language~~

~~Day 10~~

Functions and Recursion

Functions :→ In c we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}.

A function can be called multiple times to provide reusability and modularity to the c program.

Advantage of functions

(1) By using a functions, we can avoid rewriting same logic/code again and again in a program.

(2) we can call c functions any number of times in a programs and from any place in a program.

(3) we can track a large c program easily when it is divided into multiple functions.

* C Program have two types of functions

(1) Library functions

(2) User-defined functions

Library functions

C programming language has the facility to provide library functions for performing some operations. These functions are preserved in the c library and they are predefined.

For example Sqrt() is a mathematical library functions which is used for finding out the Square root of any number. The functions Scanf() and Printf() are input-output library functions.

To use library functions we have to include corresponding header file using the preprocessor directorie #include. For example to use input-output functions like Printf() and Scanf() we have to ~~include~~ include Stdio.h header file.

/* Program to find the Square root of any number */

```
#include<stdio.h>
#include<math.h>
int main()
{
    double n, s;
    Printf("Enter a number: ");
    Scanf("%lf",&n);
    s = Sqrt(n);
    Printf("The Square root is: %.2lf\n", s);
    return 0;
}
```

OUTPUT

```
Enter a number: 16
The Square root is
4.00
```

User-Defined functions

These are the functions which are created by the c programmer, so that he/she can use it many times. It reduce the complexity of a big program and optimizes the code.

a user defined functions has, three main things:

1. Functions definition
2. Functions declaration
3. Functions call.

/* program to find the Sum of two numbers */

```
#include<stdio.h>
int Sum(int x, int y);          /* Functions Declaration */
int main()
{
    int a, b, s;
    Printf("Enter Values for a and b: ");
    Scanf("%d %d",&a,&b);
    s = Sum(a, b)               /* function call */
    Printf("Sum is %d", s);
}
int Sum(int x, int y)           /* function definition */
{
    int s;
    s = x+y;
    return s;
}
```

Library functions Vs user-Defined functions

Library functions	User-Defined functions
① These functions are Predefined in the compiler of C language.	① These functions are not Pre-defined in the compiler.
② Library functions are Stored in special library file.	② user-defined functions are not Stored in library functions.
③ Execution of the program does not begin from the library function.	③ Execution of the program begins from the user define function.
④ Example ~~sum(), fact(), etc~~ Printf(), scanf(), sqrt()	④ Example ~~Printf(), Scanf(), Sqrt(), etc~~ sum(), fact() etc.

Functions Definition in C programming

A function definition in C programming language consists of function name, function parameters, return value and function body.

Syntax of function Definition

```
return_type function_name (type arg1, type arg2...)
{
    /* body of a function */
    local variable declaration;
    Statement;
    ..... ;
    return (expression);
}
```

*Key concepts

① First line is called as function header and it should be identical to function Declaration/prototype except semicolon.

② Name of arguments are compulsory here unlike function declaration.

③ Function's definition Contains code to be executed when this function is called.

return_type: The return-type is the data-type of the value returned by the function. Void data type is used when the function does not return any value. By default the return type of a function is integer (int).

function_name ⇒ This is a c identifier representing the name of the function. Every function must have a unique name in a c program. Name of a function is used to call a function.

type arg1, type arg2.. ⇒ It is a comma-separated list of data types and names of parameters passed by the calling functions. The parameter list contains data type, order and number of parameters expected by a function at the time of calling it. Parameter field is optional, we can skip it if a function don't require any data from calling function.

body of function ⇒ The body of a function contains a set of statements that will be executed when this function is called.

return(expression) This is used to send a value to calling function before termination of function. If the return type of function is Void, you need not use this line.

when control reaches return statements, it halts the execution of function immediately and returns the value of expression to calling function.

```
/* c program to show function Definition */
#include <stdio.h>
/* Function Definition */
int getAreaofSquare (int side) {
    /* Local Variable */
    int area;
    area = side * side;
    /* Return Statement */
    return area;
}
int main() {
    int side, area;
    Printf ("Enter side\n");
    Scanf ("%d", &side);
    /* calling the function */
    area = getAreaofSquare
           (side);
    Printf ("Area of Square = %d",
            area);
    return 0;
}
```

OUTPUT

```
Enter side
5
Area of Square = 25
```

Calling a Function in C Programming

The process of transferring control from one function to another, until the other function returns, is known as calling a function.

Syntax function_name (argument1, argument2, ...);

Different aspects of function

(1) Function without arguments and without return value. / No Return value and No Arguments

```
#include <stdio.h>
Void PrintName();
Void main()
{
   Print ("Hello");
   PrintName();
}
Void PrintName()
{
   Printf ("Indranil");
}
```

(2) Function without argument and with return value / with a Return value and without any argument

```
#include<stdio.h>
int Square (void);
int main()
{
   Printf ("Area of the Square\n");
   float area = Square();
   Printf ("area of Square: %f\n", area);
}
int Square()
{
   float side, square=0;
   Printf ("Enter the length of the side:\n");
   Scanf ("%f", &side);
   square= side* side;
   return square;
}
```

(3) Function with argument and without return value. / No Return Value and with Arguments

```
#include<stdio.h>
Void average (int,int,int,int,int);
Void main()
{
   int a,b,c,d,e;
   Printf ("Enter five numbers:\n");
   Scanf ("%d%d%d%d%d", &a,&b,&c,&d,&e);
   average (a,b,c,d,e);
}
Void average (int a,int b, int c,int d, int e);
{
   float avg;
   avg = (a+b+c+d+e)/5;
   Printf ("average: %f", avg);
}
```

(4) Function with argument and with return value. / Return value and with an Argument

```
#include<stdio.h>
int even_odd (int n);
{
   if (n%2==0){
      return 1;
   }else{
      return 0;
   }
}
int main() {
   int n, flag=0;
   Printf ("Enter a no\n");
   Scanf ("%d", &n);
   flag= even_odd (n);
   if (flag== 0) {
      Printf ("odd no");
   }else{
      Printf ("even no");
   }
}
```

Passing function Arguments in C Programming

- **Formal Parameter** :- This is the argument which is used inside body of function definition. It is limited to the scope of function. It gets created when control enters function and gets destroyed when control exits from function.

example

```
int sum (int x, int y ); // x and y are the formal parameters
```

- **Actual Parameter** :- This is the argument which is used while calling a function.

example

```
int main(){
    int a=10, b=20, c=0;
    c = sum (a,b); // a and b are the actual parameters
}
```

There are two methods to pass the data into the function in c language

(1) call by value (2) call by reference.

Call by Value

In call by value method, the value of the actual parameters is copied into the formal parameters.

In call by value method, we can not modify the value of the actual parameter by the formal parameter.

In call by value different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.

```
#include<stdio.h>
void swap (int, int);
void main(){
    int x=5, y=7;
    swap (x, y);
    printf("x=%d y=%d", x, y);
}
swap
void fun (int x, int y)
{
    x=7;
    y=5;
    Printf("Inside fun (called function)");
    Printf("x=%d y=%d", x, y);
}
```

Call by Reference

In call by reference, the address of the variable is passed into the function call as the actual parameter.

The value of actual parameter can be modified by changing the formal parameters since the address of actual parameters is passed.

In call by reference, the memory allocation is similar for both formal parameters and actual parameters. all the operations in the functions are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

```
#include<stdio.h>
void swap (int*, int*);
int main(){
    int a=5, b=7;
    swap(&a, &b);
    Printf("a=%d b=%d", a, b);
}
Void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    Printf("a=%d b=%d", *a, *b);
}
```

Difference between Call by Value and Call by Reference

Call by Value

(1) a copy of the value is passed into the function

(2) changes made inside the function is limited to the function only.

(3) The values of the actual parameters do not changes by changing the formal parameter.

(4) Actual and formal arguments are created at the different memory location.

Call by Reference

(1) an address of value is passed into the function.

(2) changes made inside the function validate outside of the function also.

(3) The value of actual parameters do change by changing the formal parameters.

(4) Actual and formal arguments are created at the same memory location.

Short note about main() function in C

Execution of every C program always begins with the function main(). Each function is called directly or indirectly in main() and after all function have done their operations, control return back to main(). There can be only one main() function in a program.

The main() function is a user-defined function but the name, number and type of arguments are predefined in the language. The operating system calls the main function and main() returns a value of integer type to the operating system. If the value returned is zero, it implies that the function has terminated successfully and any nonzero return value indicates an error. if no return value is specified in main() then any garbage value will be returned automatically.

Types of main() function

(1) **Void main()** :- A void is a keyword that references an empty data type that has no return value.

```
#include<stdio.h>
void main()
{
  printf("C is a programming Language");
}
```

(2) **int main()** :- An int is a keyword that references an integer data type. An int data type used with the main() functions that indicates the function should return an integer value. when we use an int main() function, it is compulsory to write return 0; statement at the end of the main() function.

```
#include<stdio.h>
int main(){
   Printf("C programming");
   return 0;
}
```

(3) int main(int argc, char *argv)

A main() function can be called using command line arguments. it is a function that contains two parameters, integer (int argc) and character (char *argv) data type. The argc parameter stands for argument count, and argv stands for argument values.

(4) int main (void) :- An int main(void) function is similar to the int main() function to return an integer value. But we can pass more than one argument to the int main(), whereas the int main(void) can only be called without any argument.

```
#include <stdio.h>
int main (void) {
    printf ("Hello world");
    return 0;
}
```

(5) Void main (void) function :- A void main (void) function is similar to the void main() function that does not return a value. However, the void main() function can accept multiple parameters, but it does not return a value. It is an empty data type, whereas void main (void) does not take any parameter because it has a predefined main (void) function.

```
#include <stdio.h>
void main (void)
{
    printf ("Hello world!");
}
```