C Programming

Lesson 3: Functions

IGATE Sensitive

Lesson Objectives

- **To understand the following topics:**
  - Storage Classes
  - Recursion
  - C variable length parameters

IGATE Sensitive

IGATE
Speed.Agility.Imagination

Module Coverage

- **In this module we will cover:**
  - Functions
    - Scope Of Variables
    - Storage Classes
    - Nesting of Functions
    - Recursion
    - Functions with variable length parameters

4.4: Function Arguments

Call by Reference

- **Instead of passing the value of a variable, the memory address (Reference) of the variable is passed to the function using Pointers. It is termed as Call by Reference**

Scope of Variables

- **The part of the program within which variable/constant can be accessed is called as its scope**
- **By default, the scope of a variable is local to the function in which it is defined**
- **Local variables can only be accessed in the function in which they are defined**
- **A variable defined outside any function is called as External variable**
- **Scope of an external variable will be the whole program, and hence such variables are referred to as Global variables**

Storage Classes

- **Storage Classes:**
  - Determine the lifetime of the storage associated with the variable.
  - When not defined for a variable, the compiler will assume a storage class depending on the context in which the variable is used
- **A variable's storage class gives the following information:**
  - Where the variable would be stored
  - What will be the default initial value
  - What is the scope of the variable
  - What is the life of the variable that is, how long would the variable exist
- **Following four types of storage classes are provided in C:**
  - Automatic Storage Class
  - Static Storage Class
  - Register Storage Class
  - External Storage Class

Automatic Storage Class

- **A variable is said to be automatic, if it is allocated storage upon entry to a segment of code, and the storage is reallocated upon exit from this segment**
- **Features of a variable with an automatic storage class are:**

Storage	Memory
Default initial value	Garbage Value
Scope	Local to the block, in which it is defined
Life	Till the control remains within the block, in which it is defined.

- **General format to declare automatic variable is:**
- **auto data-type variable-name;**
- **By default, any variable declared in a function is of the: automatic storage class;**

IGATE Sensitive

IGATE
Speed.Agility.Imagination

Automatic Storage Class (contd…)

- **They are initialized to garbage value at run-time**

```
int calculate(int rollno)
        {
                int mark, result;
        }
        are equivalent to
        int calculate(int rollno)
        {
                auto int mark, result;
        }
```

IGATE Sensitive

IGATE
Speed.Agility.Imagination

Static Storage Class

- **A variable is said to be static, if it is allocated storage at the beginning of the program execution and the storage remains allocated until the program execution terminates**
- **Variables declared outside all blocks at the same level as function definitions are always static**
- **Features of a variable with a static storage class are:**

Storage	Memory
Default initial value	Zero
Scope	Local to the block, in which it is defined
Life	Value of the variable persists between different function calls.

- **General format for declaring static variable is:**
- **static data-type variable-name;**

Register Storage Class

- **In cases when faster computation is required, variables can be placed in the CPU's internal registers, as accessing internal registers take much less time than accessing memory**
- **Therefore, if a variable is used at many places in a program it is better to declare its storage class as register**
- **Only character and integer variables are supported**
- **But, the compiler will decide whether a variable is to be stored as a register variable or automatic**

Register Storage Class (contd…)

- **Features of a variable with a register storage class are:**

Storage	CPU registers
Default initial value	Garbage Value
Scope	Local to the block, in which it is defined
Life	Till the control remains within the block, in which it is defined.

- **General format of declaring register variable is:**
- **register data-type variable-name;**

IGATE Sensitive

IGATE Speed.Agility.Imagination

External Storage Class

- **If the declared variable is needed in another file, or in the same file but at a point earlier than that at which it has been defined, it must be declared of storage class external**
- **Features of a variable with an external storage class are:**

Storage	Memory
Default initial value	Zero
Scope	Global
Life	As long as the program's execution doesn't come to an end

- **An extern data type variable declaration is of the form:**
- **extern date-type variable-name;**
- **Definition of an external variable, specified without the keyword extern, causes the storage to be allocated, and also serves as the declaration for the rest of that source file**

External Storage Class: Example

- **Following example shows the definition, declaration and use of external variables**

main.c	**compute.c**

#include <stdio.h>
int calculate(void);
/* Declaration of Theory_Mark & Practical Mark */
int Theory_Mark,Practical_Mark;
int main(void)
{printf(“Enter Marks”);
scanf(“%d%d”,&Theory_Mark,&Practical_Mark);
printf(“Module Mark=%d”,calculate());
return 0;
}
	#include <stdio.h>
	/* Extern declaration of Theory_Mark & * Practical_Mark variables.
	* No new variables are created */
	extern int Theory_Mark,Practical_Mark;
	int output(void); int result=0;
	int calculate()
	{result=Theory_Mark+Practical_Mark;
	return(result);
	}

Which to use and When

- **Tips for usage of different storage classes in different situations:**
  - Static storage class is used only if you want the value of a variable to persist between different function calls. Typical application of this storage is recursive functions
  - Register storage class is used only those variables which are being used very often in the program. Typical application of this storage is loop counters, which get used a number of times in a program since the access is faster
  - Extern storage class is used only those variables which are being used by almost all the functions in the program. This would avoid unnecessary passing of these variables as arguments when making a function call. If you don’t have any of the express needs mentioned above, then use the auto storage class

Recursion

- Recursion is a process by which a function calls itself repeatedly, until some specified condition has been satisfied
- The process is used for repetitive computations in which each action is stated in terms of a previous result
- Functions may be defined recursively; that is, a function may directly or indirectly call itself in the course of execution
- If the call to a function occurs inside the function itself, the recursion is said to be direct
- If a function calls another function, which in turn makes a call to the first one, the recursion is said to be indirect
- The chain of calls may be more involved; there may be several intermediate calls before the original function is called back

Recursion (contd…)

- **For example:**

```
long int factorial(int fact_num)
{
        if((fact_num == 0)||(fact_num == 1))
                return 1;
        else
                return(fact_num * factorial(fact_num-1));
}
```

Recursion: Example

- **Code Snippet**

```
/*To print fibonacci series of first 10 numbers using recursion */
# include <stdio.h>
int fibonacci(int);
int fibonacci(int index_value)
{
        if(index_value==1||index_value==2)
           return 1;
        else
           return(fibonacci(index_value-1)+fibonacci(index_value-2));
}
```

Recursion: Example (Contd..)

```
void main(void)
{
        int index_value;
        for(index_value=1; index_value <=10; index_value ++)
        printf(“%d\n”,fibonacci(index_value));
}
```

Functions

- **Function names should be meaningful**
- **Function names should reflect what they do and what they return**
  - Functions are used in expressions, often in an if clause, so they need to read appropriately. For example:
    - if (checksize(x))
      - The above expression is unhelpful because it does not tell us whether checksize returns true on error or non-error
    - if (validsize(x))
      - Instead the above expression makes the point clear
  - By making function names verbs and following other naming conventions programs can be read more naturally
    - Good example:

      check_for_errors()
    - Bad example:

      error_check()

Functions (contd…)

- **scanf should never be used in serious applications. Its error detection is inadequate. Look at the example below:**

```
int main(void)
{ int i; float f;
 printf("Enter an integer and a
float: ");
scanf("%d %f", &i, &f);
printf("I read %d and %f\n", i, f);
 return 0;
}
```

Test run

```
Enter an integer and a float: 182 52.38
Output: 182 and 52.380001
```

Another TEST run

```
Enter an integer and a float: 6713247896
4.4
Output: -1876686696 and 4.400000
```

IGATE Sensitive

Functions (contd…)

- **getchar() - macro or function:**
  - The following program copies its input to its output:

```
#include  <stdio.h>
int main(void)   {
register int empno;
while ((empno = getchar()) != EOF)
putchar(empno);              }
```

  - Removing the #include statement from the program would cause it to fail to compile because EOF would then be undefined

Functions (contd…)

– We can rewrite the program in the following way:

```
#define EOF -1
int main(void)
{
        register int empno;
        while ((empno = getchar()) != EOF)
            putchar(empno);
}
```

– This will work on many systems but on some it will run much more slowly

IGATE Sensitive

IGATE Speed.Agility.Imagination

Functions (contd…)

- **Avoid Defining functions before main as the reader have to search for main, which is the place to start when analyzing code and it gets annoying**
- **Hence it reduces the readability**

```
int tax_calculation();
int main()
{ tax_calculation();
 return 13; }
int tax_calculation()
{ // ... ten pages worth of code. }
```

- **If a program is making use of multiple functions then it is a good practice to keep those functions in separate file**
- **It increase the performance and will be easy to debug**

C variable length parameters

- Variable length parameters is for function like printf(). The length of parameters is not fixed for printf(). We can pass in as many parameters as we want.
- For example,

```
sum();
sum(1);
sum(1,2);
sum(1,2,3);
```

- The sum function just adds all parameters together.
- **The list of Functions to use for this are :**

```
type va_arg(va_list argptr, type);
void va_copy(va_list target, va_list source);
void va_end(va_list argptr);
void va_start(va_list argptr, last_parm);
```

- We use va_arg(), va_start(), and va_end() to deal with a variable number of arguments to a function.

Lab Session

- **Lab 3**

Summary

- **In this lesson, you have learnt:**
  - Functions break large complicated computing tasks into smaller and simpler ones .
  - The functions written by the user, are termed as “User Defined Functions”.
  - If a variable is defined outside any function at the same level as function definitions is called as External variable.
  - Using variable length of arguments utility one can pass in as many parameters as they want.

IGATE Sensitive

IGATE Speed.Agility.Imagination

Review Question

- **Complete the following statement:**
  - Function declaration is also called as __________.

- **True or False?**
  1. If return type of function is not specified, it defaults to int. (True / False)

  2. A C program consists of one or more functions with one main() function. (True / False)

IGATE Sensitive

Review Question: Match the Following

1. Automatic Storage class	1. Value of the variable persists between different function calls.
2. Static storage class	2. Till the control remains within the block, in which it is defined.
3. Register storage class	3. As long as the program’s execution doesn’t come to an end.
4. External storage class	4. Is typically specified for heavily used variables enhancing performance by minimizing access time.

IGATE Sensitive