RV College of Engineering®

Go, change the world

Course Name: Programming in C

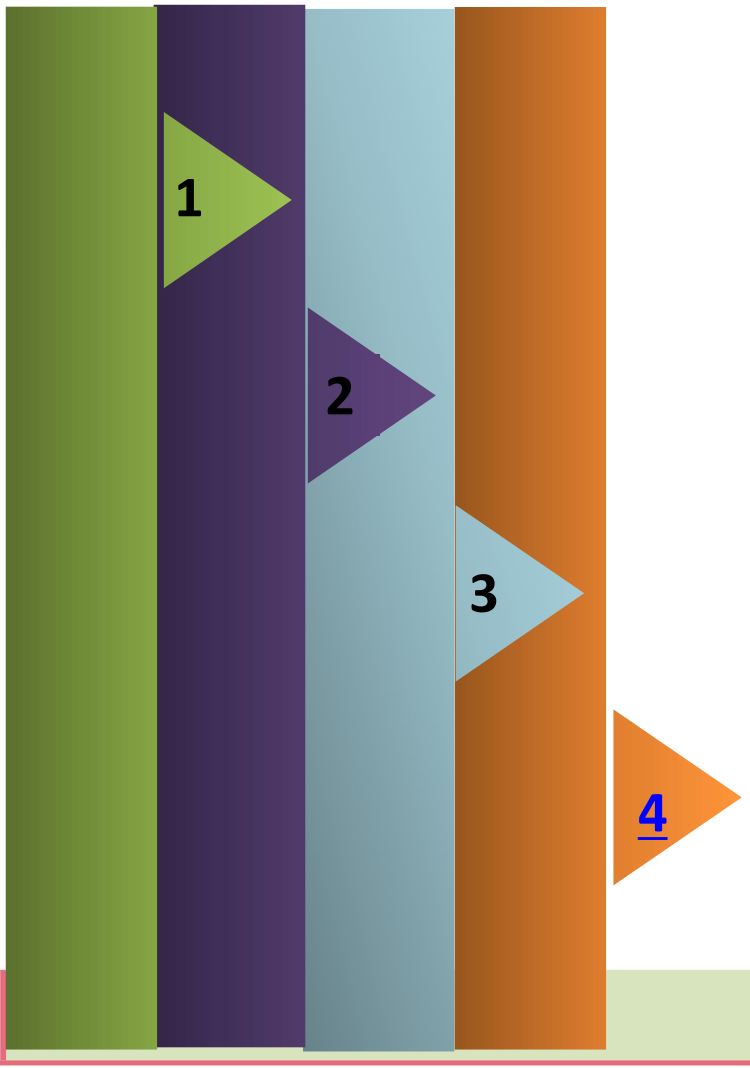
Course Code: 21CS13

Unit 4A- Functions and Storage Classes

Prof. Veena Gadad

Department of Computer Science and Engineering

RVCE



Contents

Introduction

- The structured programming approach to program design was based on the following methods:
  - Solving a large problem, by breaking it into several pieces and working on each piece separately
  - Solving each piece by treating it as a new problem that can itself be broken down into smaller problems
  - Repeating the process with each new piece until each can be solved directly, without further decomposition

Concept of function

- A function is a self-contained block of program statements that performs a particular task.
- It is often defined as a section of a program performing a specific job.
- The separate program segment is called a function and the program that calls it is called the ‘main program’.

Need of a function

1. It makes programs significantly easier to **understand and maintain** by breaking up a program into easily manageable chunks.
2. Reduces the lines of code in the main program.
3. Code reusability- The C standard library is an example of the reuse of functions. This enables code sharing.
4. Protect the data.

Using function

- All C programs contain at least one function, called main() where execution starts.
- Returning from this function, the program execution terminates and the returned value is treated as an indication of success or failure of program execution.
- When a function is called, the code contained in that function is executed, and when the function has finished executing, control returns to the point at which that function was called.
- Functions are used by calling them from other functions. When a function is used, it is referred to as the 'called function'.

Using function

- The functions have three components:

1. Function prototype declaration.
2. Function Call.
3. Function Definition.

Function prototype declaration.

- A user-written function should normally be declared prior to its use to allow the compiler to perform type checking on the arguments used in its call statement or calling construct.
- The general form of this function declaration statement is as follows:

```
return_data_type function_name (data_type variable1, ...);
```

- **function_name** This is the name given to the function and it follows the same naming rules as that for any valid variable in C.
- **return_data_type** : This specifies the type of data given back to the calling construct by the function after it executes its specific task.
- **data_type_list** This list specifies the data type of each of the variables, the values of which are expected to be transmitted by the calling construct to the function

Function prototype declaration.

The following are some examples of declaration statements.

(a) float FtoC(float faren);

(b) double power(double, int);

(c) int isPrime(int);

(d) void printMessage(void);

(e) void fibo_series(int);

Function Definition.

- The collection of program statements in C that describes the specific task done by the function is called a function definition.
- It consists of the **function header** and a **function body**, which is a block of code enclosed in parentheses.
- The definition creates the actual function in memory.
- The general form of the function definition is as follows:

```
return_data_type function name(data_type variable1, data_type variable2,……)
  {
                        /* Function Body */
  }
```

Function Header

Function Body

Function Definition.

- **Function header** is similar to the function declaration, but does not require the semicolon at the end.
- The list of variables in the function header is also referred to as the **formal parameters.**

Write a function that computes x^n , where x is any valid number and n an integer value

Function Definition.

Write a function that computes x^n , where x is any valid number and n an integer value

```
double power(double x, int n)          /* function header */
{
                                           /* function body starts here... */
 double result = 1.0;                     /* declaration of variable result */
 for(int i = 1; i<=n; i++)
result *= x;                              /* : */
return result;                           /* return value in 'result' to calling function*/
}                                          /* function body ends here... */
```

Function Definition.

Function for converting a temperature from Fahrenheit scale to Celsius scale.

```
float FtoC(float faren)          /*function header */
{                                /* function body starts here…….*/
float factor = 5.0/9.0;
float freezing = 32.0;
float celsius;
celsius = factor *(faren - freezing);
return celsius;
}                                /* function body ends here……. */
```

Function Definition.

NOTE:

1. A function can only return one value.
2. A function with return type void does not return any value.
3. There may be more than one return statement in a function, but only one return statement will be executed per call to the function.

Function Definition.

Function definition to check whether a given year is a leap year or not

```
void leap_yr(int yr)
{
    if((yr%4==0)&&(yr%100!=0)||yr%400 ==0)
        return 1;
    else
        return 0;
}
```

Function Call.

- A function will carry out its expected action whenever it is invoked (i.e. whenever the function is called) from some portion of a program.
- The program control passes to that of the called function.
- Once the function completes its task, the program control is returned back to the calling function.
- **The general form of the function call statement (or construct) is:**

```
function_name(variable1, variable2,…);
```

or

```
variable_name = function_name(variable1, variable2,…);
```

```
function_name();
```

or

```
variable_name = function_name();
```

❖ *Information will be passed to the function via special identifiers or expression called arguments or actual parameters*

Function Declaration, Definition and Call


```
#include<stdio.h>
int  fact(int n);
int main()
{
....
Z= fact(int n);
....
}
```

```
int fact(int n)
{
......
......
return  result;
}
```

1. Function Declaration.
2. Function Call.
3. Function Definition.
4. Arguments
5. Parameters

Function Definition.

Function for converting a temperature from Fahrenheit scale to Celsius scale.

```
#include <stdio.h>
float FtoC(float); //Function prototype declaration
int main(void)
{
float tempInF;
float tempInC;
printf("\n Temperature in Fahrenheit scale: ");
scanf("%f", &tempInF);
tempInC = FtoC(tempInF);  //Function calling

printf("%f Fahrenheit equals %f Celsius \n",
tempInF,tempInC);
 return 0;
}
```

```
float FtoC(float faren) /*function header */
{                                  /* function body starts here…….*/
float factor = 5.0/9.0;
float freezing = 32.0;
float celsius;
celsius = factor *(faren - freezing);
return celsius;
}                                  /* function body ends
here……. */
```

Function Call Mechanisms

- The technique used to pass data to a function is known as parameter passing.
- Data is passed to a function using one of the two techniques: pass by value or call by value and pass by reference or call by reference.
- In **call by value**, a copy of the data is made and the copy is sent to the function. The copies of the value held by the arguments are passed by the function call.
- In **call by reference**, sends the address of the data rather than a copy. In this case, the called function can change the original data in the calling function.

Function Call Mechanisms

- In **call by value**, a copy of the data is made and the copy is sent to the function. The copies of the value held by the arguments are passed by the function call.
- Since only copies of values held in the arguments are passed by the function call to the formal parameters of the called function, the value in the arguments remains unchanged.
- As only copies of the values held in the arguments are sent to the formal parameters, the function cannot directly modify the arguments passed.

Function Call Mechanisms


```
#include int mul_by_10(int num); /* function prototype */
int main(void)
 {
   int result, num = 3;
   printf("\n num = %d before function call.", num);
   result = mul_by_10(num);
   printf("\n result = %d after return from function", result);
   printf("\n num = %d", num);
   return 0;
}
/* function definition follows */
int mul_by_10(int num)
 {
      num *= 10;
      return num;
 }
```

num = 3, before function call.
result = 30, after return from function.
num = 3

Function Call Mechanisms

- In **call by reference**, sends the address of the data rather than a copy. In this case, the called function can change the original data in the calling function.
- For example: Passing arrays to functions.
  - When an array is passed to a function, the address of the array is passed and not the copy of the complete array.

Function Call Mechanisms


```
#include <stdio.h>
int maximum(int [],int); /* function prototype */
int main(void)
{
int values[5], i, max;
printf("Enter 5 numbers\n");
for(i = 0; i < 5; ++i)
scanf("%d", &values[i]);
max = maximum(values,5); /* function call */
printf("\nMaximum value is %d\n", max);
 return 0;
}
```

```
/**** function definition ****/
int maximum(int values[], int n)
{
 int max_value, i;
 max_value = values[0];
 for(i = 1; i < n; ++i)
 if(values[i] > max_value)
 max_value = values[i];
 return max_value;
}
```

Output

Enter 5 numbers

11 15 8 21 7

Maximum value is 21

Function Call Mechanisms

Write a program that uses a function to perform addition and subtraction of two matrices having integer numbers.

```
#include <stdio.h>
#define row 2
#define col 3
void mat_arith(int [][col], int [][col]);
/* function prototype */
int main()
{
 int a[row][col], b[row][col],i,j;
printf("\n Enter elements of the first
matrix.\n");
 for(i=0; i<row; i++)
/** Read first matrix elements **/
 for(j=0; j<col; j++)
scanf("%d",&a[i][j]);
printf("\n Enter elements of the second
matrix.\n");
 for(i=0; i<row; i++)
/** Read second matrix elements **/
 for(j=0; j<col; j++)
scanf("%d",&b[i][j]);
 mat_arith(a,b); /** function call **/
}
void mat_arith(int a[][col], int b[][col])
{
 int c[row][col],i,j,choice;
printf("\n For addition enter: 1 \n")
printf("For subtraction enter: 2\n");
printf("\nEnter your choice:");
scanf("%d",&choice);
for(i=0; i<row; i++)
 for(j=0; j<col; j++)
 {
if(choice==1)
 c[i][j]= a[i][j] + b[i][j];
 else if(choice==2)
 c[i][j]= a[i][j] - b[i][j];
 else
 {
printf("\n Invalid choice. Task not done.");
 return;
 }
 }
printf("\n The resulting matrix is:\n");
 for(i=0; i<row; i++)
 {
 for(j=0; j<col; j++)
printf("%d", c[i][j]);
printf("\n\n");
 }
 return;
}
```

Storage Classes in C

- In C, the variables are declared by the type of data they can hold.
- During the execution of the program, these variables may be stored in the CPU registers or the primary memory of the computer.
- To indicate where the variables would be stored, how long they would exist, what would be their region of existence, and what would be the default values, C provides four storage class specifiers that can be used along with the data type specifiers in the declaration statement of a variable.
- These four storage class specifiers are **automatic, external, register, and static**

Storage Classes in C

- The storage class specifier precedes the declaration statement for a variable.
- The general form of the variable declaration statement that includes the storage class specifier is given as follows:

storage_class_specifier data_type variable_name;

Storage Class Auto

- By default, all variables declared within the body of any function are automatic.
- The keyword auto is used in the declaration of a variable to explicitly specify its storage class.
- For example, the following declaration statement within a function body

```
auto char any_alpha;
```

- **These variables are stored in the primary memory of the computer**

Storage Class Register

- Values stored in registers of the CPU are accessed in much lesser time than those stored in the primary memory.
- To allow the fastest access time for variables, the register storage class specifier is used.
- The keyword for this storage class is register.

For example: register int a;

Storage Class Register

NOTE:

1. Global variables with register storage class are not allowed.
2. In C, it is not possible to obtain the address of a register variable by using '&' operator.
3. In addition, the only storage class specifier that can be used in a parameter declaration is register

Storage Class Static

- Two types of variables are allowed to be specified as static variables: **local variables and global variables.**
- The local variables are also referred to *as internal static variables* whereas the global variables are also known as *external static variables*.
- The default value of a static variable is zero.
- To specify a local variable as static, the keyword static precedes its declaration statement

Storage Class Static

- A static local variable is allotted a permanent storage location in the primary memory.
- This variable is usable within functions or blocks where it is declared and preserves its previous value held by it between function calls or between block re-entries.
- However, once a function is invoked, the static local variable retains the value in it and exists as long as the main program is in execution.

Storage Class Static

- The external static variables in a program file are declared like global variables with the keyword static preceding its declaration statement.
- These static variables are accessible by all functions in the program file where these variables exist and are declared.

Storage Class Static


```
#include int main()
{
void show(void);
 printf("\n First Call of show()");
 show();
printf("\n Second Call of show()");
show();
printf("\n Third Call of show()");
 show();
 return 0;
 }
```

```
void show(void) {
static int i;
printf("\n i=%d",i);
 i++;
}
```

Output

First Call of show() i=0

Second Call of show() i=1

Third Call of show() i=2

Storage Class Extern

- A program in C, particularly when it is large, can be broken down into smaller programs.
- After compiling, each program file can be joined together to form the large program.
- These small program modules that combine together may need some variables that are used by all of them.
- These variables are global to all the small program modules that are formed as separate files.
- The keyword for declaring such global variables is extern.

Storage Class Extern


```
//first.c
extern int i;
/***** function definition of show()*********/
void show(void) /*** function header ***/
{ /*** fn. body starts..**/
printf("\n Value of i in pgm2.c=%d",i);
}
//second.c
#include <stdio.h>
#include "first.c" /*** link program pgm2.c ***/
int i; /*** external/global decl.**/
void show(void); /*** function prototype ***/
int main()
{
 i=10;
show(); /* call to function in program file
 pgm2.c */
printf("\nValue of i in pgm1.c=%d",i);
 return 0;
}
```

Storage Class Summary

Storage class specifier	Place of storage	Scope	Lifetime	Default value
`auto`	Primary memory	Within the block or function where it is declared.	Exists from the time of entry in the function or block to its return to the calling function or to the end of block.	garbage
`register`	Register of CPU	Within the block or function where it is declared.	Exists from the time of entry in the function or block to its return to the calling function or to the end of block.	garbage
`static`	Primary memory	***For local*** Within the block or function where it is declared. ***For global*** Accessible within the combination of program modules/files that form the full program.	***For local*** Retains the value of the variable from one entry of the block or function to the next or next call. ***For global*** Preserves value in the program file.	0
`extern`	Primary memory	Accessible within the combination of program modules/files that form the full program.	Exists as long as the program is in execution.	0