

INTRODUCTION TO PROGRAMMING LAB MANUAL

SUBJECT CODE- UCS02P08

Computer Science & Engineering Department



राष्ट्रीय प्रौद्योगिकी संस्थान अगरतला
National Institute of Technology Agartala
Department of Computer Science and Engineering
Agartala, Jirania– 799046

Fundamental C Programs

Basic Discussion Topics:

- ✓ What is file extension
- ✓ What is file extension for c programming for source code
- ✓ What is file extension for c ++ programming for source code
- ✓ What is the step to run a program successfully .
- ✓ What is logical error and syntax error.
- ✓ Naming conventions for defining a variable.
- ✓ Say yes or no
 - Can we use the clrscr() function before the declaration statement ?
 - Can we use the clrscr() function for multiple times ?
 - Can we use getch() function in any place of the source code within void main() function?
- ✓ What is the actual function of the getch(), clrscr(), printf(), scanf() and their required header file?
- ✓ How many types of the file are being created during compiling a source code & where it is stored?
- ✓ Define interpreter, Compiler & their basic differences.
- ✓ What is the Starting point Of a C program?
- ✓ Requirement of an ASCII chart in a C program?
- ✓ What are the Reserve Word & how Many Reserve word are available in C
- ✓ Why we call that C is a free-form Language?
- ✓ What is the Format Specifier ? What is the function of \n,\a,\b\t ?

Basic C Programming exercises

- 1) C "Hello, World!" Program
- 2) C program for Input/output of Integer, Character and Floating point numbers C Program to Add Two Integers
- 3) C Program to Multiply two Floating Point Numbers
- 4) C Program to Find ASCII Value of a Character
- 5) C Program to Compute Quotient and Remainder
- 6) C Program to Find the Size of int, float, double and char
- 7) C Program to Swap Two Numbers using and without using third variable
- 8) C Program to find area of a triangle
- 9) C Program to find area of a circle
- 10) C Program to find area of a Square
- 11) C program to find square root of a number using sqrt function
- 12) C Program to calculate gross salary of a person.
Where DA is 12% of basic Salary, HR is 7% of basic Salary and TA is 5% of Basic Salary.

Decision Making Statements or Conditional Statements

C if...else Statement

In programming, decision making is used to specify the order in which statements are executed. In this tutorial, you will learn to create decision making program using if...else statements.

C if statement

```
if (testExpression)
{
    // statements
}
```

The if statement evaluates the test expression inside the parenthesis.

If the test expression is evaluated to true (nonzero), statements inside the body of if is executed.

If the test expression is evaluated to false (0), statements inside the body of if is skipped from execution.

Flowchart of if statement

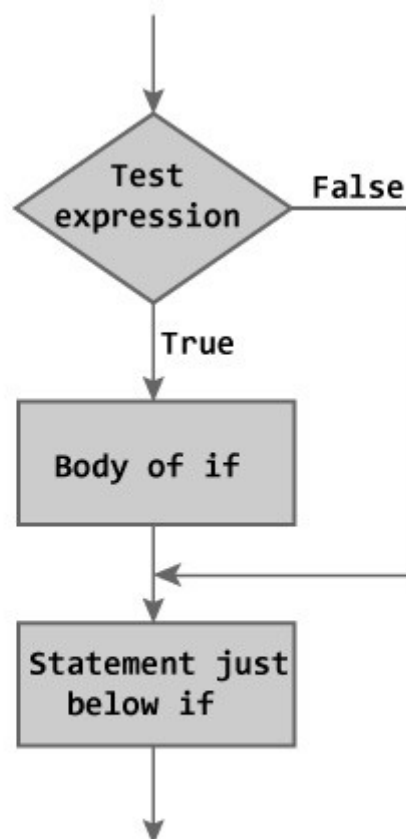


Figure: Flowchart of if Statement

Programming exercises

- 1) C Program to Check Whether a Number is Even or Odd
- 2) C Program to Check Whether a Character is Vowel or Consonant
- 3) C Program to Find the Largest Number Among Two Numbers
- 4) C Program to Find the Largest Number Among Three Numbers
- 5) C Program to Check Leap Year.
- 6) Write a C program to check whether a character is uppercase or lowercase alphabet.
- 7) Write a C program to input week number print week day.
- 8) Write a C program to input month number and print number of days in that month.
- 9) Write a C program to find all roots of a quadratic equation.
- 10) Write a C program to calculate profit or loss.

- 11) Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:
Percentage $\geq 90\%$: Grade A
Percentage $\geq 80\%$: Grade B
Percentage $\geq 70\%$: Grade C
Percentage $\geq 60\%$: Grade D
Percentage $\geq 40\%$: Grade E
Percentage $< 40\%$: Grade F
- 12) Write a C program to input basic salary of an employee and calculate its Gross salary according to following:
Basic Salary ≤ 10000 : HRA = 20%, DA = 80%
Basic Salary ≤ 20000 : HRA = 25%, DA = 90%
Basic Salary > 20000 : HRA = 30%, DA = 95%
- 13) Write a C program to input electricity unit charges and calculate total electricity bill according to the given condition:
For first 50 units Rs. 0.50/unit
For next 100 units Rs. 0.75/unit
For next 100 units Rs. 1.20/unit
For unit above 250 Rs. 1.50/unit
An additional surcharge of 20% is added to the bill

C Ternary Operator (?:)

A conditional operator is a ternary operator, that is, it works on 3 operands.

Conditional Operator Syntax

`conditionalExpression ? expression1 : expression2`

The conditional operator works as follows:

The first expression `conditionalExpression` is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.

If `conditionalExpression` is true, `expression1` is evaluated.

If `conditionalExpression` is false, `expression2` is evaluated.

Programming exercises

1. Write a C program to find maximum between two numbers using conditional operator.
2. Write a C program to find maximum between three numbers using conditional operator.
3. Write a C program to check whether a number is even or odd using conditional operator.
4. Write a C program to check whether year is leap year or not using conditional operator.
5. Write a C program to check whether character is an alphabet or not using conditional operator.

(Ref: <http://codeforwin.org/2015/06/conditional-operator-programming-exercise.html>)

LOOPS

Loops are used in programming to repeat a specific block of code. After reading this tutorial, you will learn to create a for loop in C programming. Loops are used in programming to repeat a specific block until some end condition is met. There are three loops in C programming:

- for loop
- while loop
- do...while loop

C Programming for Loop

The syntax of for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
    // codes
}
```

How for loop works?

The initialization statement is executed only once.

Then, the test expression is evaluated. If the test expression is false (0), for loop is terminated. But if the test expression is true (nonzero), codes inside the body of for loop is executed and the update expression is updated.

This process repeats until the test expression is false.

The for loop is commonly used when the number of iterations is known.

For loop Flowchart

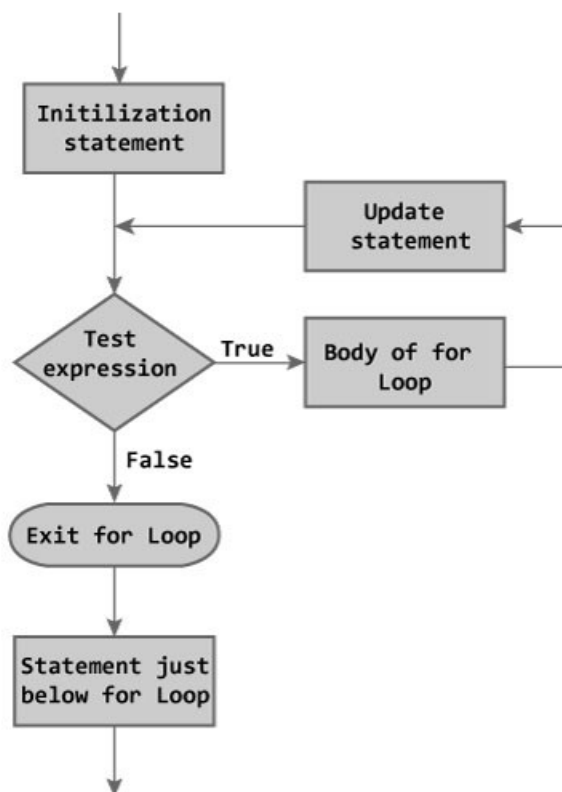


Figure: Flowchart of for Loop

Programming exercises

- 1) C Program to print all the number up to n
- 2) C Program to Calculate the Sum of Natural Numbers
- 3) C Program to print all the even number up to n
- 4) C Program to print all the odd number up to n
- 5) C Program to print sum of all even & odd numbers up to n
- 6) C Program to Find Factorial of a Number
- 7) C Program to Generate Multiplication Table
- 8) C Program to Display Fibonacci Sequence
- 9) C Program to print list of all Prime Number up to a certain Limit.
- 10) C Program to find sum of prime numbers
- 11) C program to print right triangle star pattern series

```
  *
 * *
* * *
* * * *
```

- 13) C program to print equilateral triangle or pyramid star pattern

```
  *
 * *
* * * *
* * * * *
```

- 14) C program to print Floyd's triangle

```
1
2 3
4 5 6
7 8 9 10
```


While & do while Loop

The while loop evaluates the test expression.

If the test expression is true (nonzero), codes inside the body of while loop are executed. The test expression is evaluated again. The process goes on until the test expression is false.

When the test expression is false, the while loop is terminated.

Flowchart of while loop

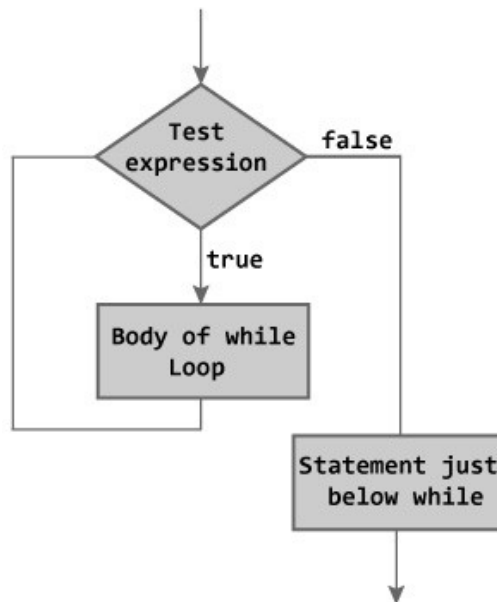


Figure: Flowchart of while Loop

The `do...while` loop is similar to the `while` loop with one important difference. The body of `do...while` loop is executed once, before checking the test expression. Hence, the `do...while` loop is executed at least once.

How do...while loop works?

The code block (loop body) inside the braces is executed once.

Then, the test expression is evaluated. If the test expression is true, the loop body is executed again. This process goes on until the test expression is evaluated to 0 (false).

When the test expression is false (nonzero), the `do...while` loop is terminated.

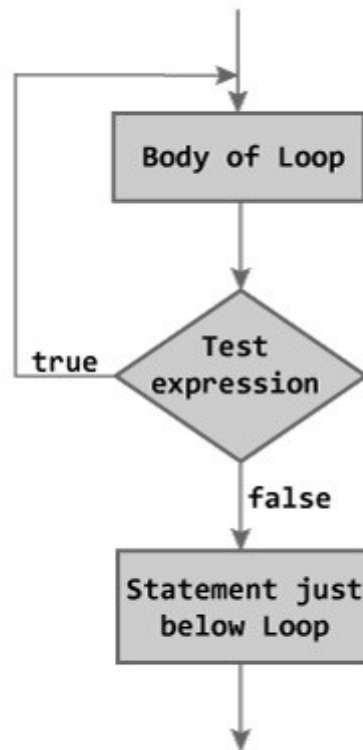


Figure: Flowchart of do...while Loop

Programming exercises

- 1) Program to print all the number up to n
- 2) C Program to Calculate the Sum of Natural Numbers
- 3) C Program to print all the even number up to n
- 4) C Program to print all the odd number up to n
- 5) C Program to print sum of all even & odd numbers up to n
- 6) C Program to Find Factorial of a Number
- 7) C Program to Generate Multiplication Table
- 8) C Program to Display Fibonacci Sequence
- 9) C Program to print list of all Prime Number up to a certain Limit.
- 10) C Program to find sum of prime numbers
- 11) C Program to Count Number of Digits in an Integer
- 12) C Program to Reverse a Number
- 13) C Program to Calculate the Power of a Number
- 14) C Program to Check Whether a Number is Palindrome or Not
- 15) C Program to Check Armstrong Number
- 16) C Program to Display Factors of a Number

C switch...case Statement

The `if..else..if` ladder allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in `if...else...if`, it is better to use `switch` statement.

The `switch` statement is often faster than nested `if...else` (not always). Also, the syntax of `switch` statement is cleaner and easy to understand.

```
switch (n)
{
    case constant1:
        // code to be executed if n is equal to constant1;
        break;

    case constant2:
        // code to be executed if n is equal to constant2;
        break;
    .
    .
    .
    default:
        // code to be executed if n doesn't match any constant
}
```

When a case constant is found that matches the switch expression, control of the program passes to the block of code associated with that case.

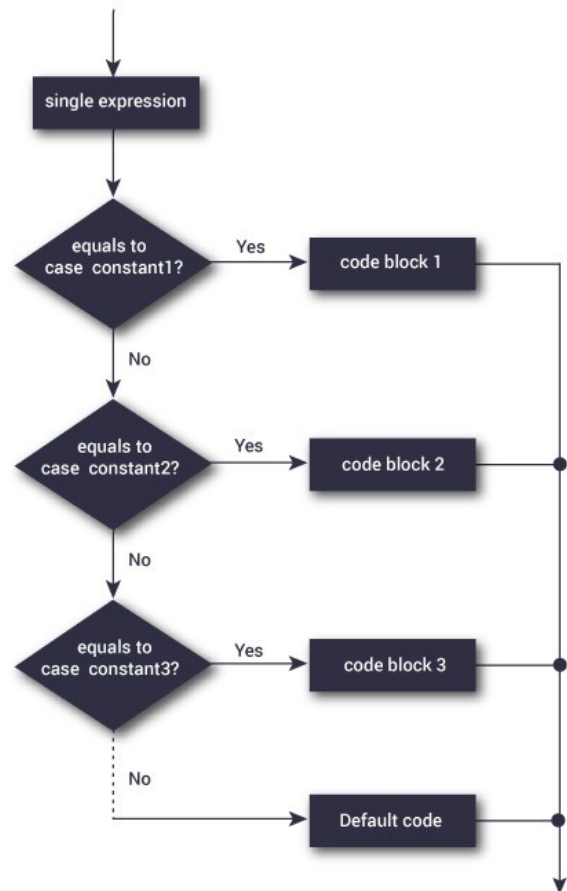
In the above pseudo code, suppose the value of `n` is equal to `constant2`. The compiler will execute the block of code associated with the case statement until the end of switch block, or until the `break` statement is encountered.

The `break` statement is used to prevent the code running into the next case.

Programming exercises

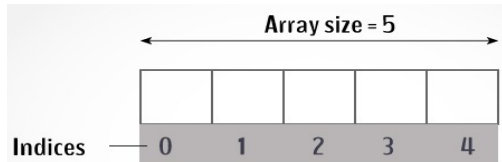
Switch Statement Flowchart

- 1) C Program to find whether a character is vowel or consonant.
- 2) C Program to Make a Simple calculator.
- 3) C program to check whether number is EVEN or ODD using switch.
- 4) C program to find number of days in a month using switch case.
- 5) C program to read weekday number and print weekday name using switch.



C Programming Arrays

An array is a collection of data that holds fixed number of values of same type. For example: if you want to store marks of 100 students, you can create an array for it.



```
float marks[100];
```

The size and type of arrays cannot be changed after its declaration.

Arrays are of two types:

1. One-dimensional arrays
2. Multidimensional arrays (will be discussed in next chapter)

How to declare an array in C?

```
data_type array_name[array_size];
```

For example,

```
float mark[5];
```

Here, we declared an array, `mark`, of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

Elements of an Array and How to access them?

You can access elements of an array by indices.

Suppose you declared an array `mark` as above. The first element is `mark[0]`, second element is `mark[1]` and so on.

<code>mark[0]</code>	<code>mark[1]</code>	<code>mark[2]</code>	<code>mark[3]</code>	<code>mark[4]</code>

Few key notes:

- Arrays have 0 as the first index not 1. In this example, `mark[0]`
- If the size of an array is `n`, to access the last element, `(n-1)` index is used. In this example, `mark[4]`
- Suppose the starting address of `mark[0]` is 2120d. Then, the next address, `a[1]`, will be 2124d, address of `a[2]` will be 2128d and so on. It's because the size of a float is 4 bytes.

\

How to initialize an array in C programming?

It's possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

Another method to initialize array during declaration:

```
int mark[] = {19, 10, 8, 17, 9};
```

<code>mark[0]</code>	<code>mark[1]</code>	<code>mark[2]</code>	<code>mark[3]</code>	<code>mark[4]</code>
19	10	8	17	9

Here,

```
mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9
```

How to insert and print array elements?

```
int mark[5] = {19, 10, 8, 17, 9}

// insert different value to third element
mark[3] = 9;

// take input from the user and insert in third element
scanf("%d", &mark[2]);

// take input from the user and insert in (i+1)th element
scanf("%d", &mark[i]);

// print first element of an array
printf("%d", mark[0]);

// print ith element of an array
printf("%d", mark[i-1]);
```

Programming exercises

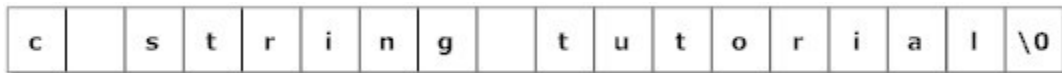
- 1) C Program to Calculate Average of n ($n < 10$) numbers using arrays
- 2) C Program to Find Largest Element of an Array
- 3) Write a program in C to find the maximum and minimum element in an array.
- 4) Write a program in C to separate odd and even integers in separate arrays.
- 5) Write a program in C to sort elements of array in ascending order.
- 6) C Program to Add Two Matrix Using Multi-dimensional Arrays
- 7) C Program to multiply Two Matrix Using Multi-dimensional Arrays

C Programming Strings

In C programming, array of characters is called a string. A string is terminated by a null character `/0`. For example:

```
"c string tutorial"
```

Here, "c string tutorial" is a string. When, compiler encounter strings, it appends a null character `/0` at the end of string.



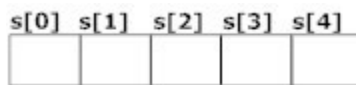
Declaration of strings

Before we actually work with strings, we need to declare them first.

Strings are declared in a similar manner as arrays. Only difference is that, strings are of `char` [type](#).

Using arrays

```
char s[5];
```



Using pointers

Strings can also be declared using [pointer](#).

```
char *p;
```

Initialization of strings

In C, string can be initialized in a number of different ways.

For convenience and ease, both initialization and declaration are done in the same step.

Using arrays

```
char c[] = "abcd";  
OR,  
char c[50] = "abcd";  
OR,  
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

```
OR,  
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```

c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

The given string is initialized and stored in the form of arrays as above.

Using pointers

String can also be initialized using pointers as:

```
char *c = "abcd";
```

Reading Strings from user

You can use the `scanf()` function to read a string like any other data types.

However, the `scanf()` function only takes the first entered word. The function terminates when it encounters a white space (or just space).

Reading words from user

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

Example #1: Using `scanf()` to read a string

```
#include <stdio.h>  
int main()  
{  
    char name[20];  
    printf("Enter name: ");  
    scanf("%s", name);  
    printf("Your name is %s.", name);  
    return 0;  
}
```

Output

```
Enter name: Dennis Ritchie  
Your name is Dennis.
```


Here, program ignores Ritchie because, scanf() function takes only a single string before the white space, i.e. Dennis.

Reading a line of text

An approach to reading a full line of text is to read and store each character one by one.

Example #2: Using getchar() to read a line of text

```
#include <stdio.h>
int main()
{
    char name[30], ch;
    int i = 0;
    printf("Enter name: ");
    while(ch != '\n') // terminates if user hit enter
    {
        ch = getchar();
        name[i] = ch;
        i++;
    }
    name[i] = '\0'; // inserting null character at end
    printf("Name: %s", name);
    return 0;
}
```

In the program above, using the function getchar(), ch gets a single character from the user each time.

This process is repeated until the user enters return (enter key). Finally, the null character is inserted at the end to make it a string.

This process to take string is tedious.

Example #3: Using standard library function to read a line of text

2. C program to read line of text using gets() and puts()

To make life easier, there are predefined functions gets() and puts in C language to read and display string respectively.

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    gets(name); //Function to read string from user.
    printf("Name: ");
    puts(name); //Function to display string.
    return 0;
}
```

Both programs have the same output below:

Output

```
Enter name: Tom Hanks  
Name: Tom Hanks
```

Programming exercises

- 1) C Program to Reverse a String
- 2) C Program to Find the Length of a String
- 3) C program to Concatenate Two Strings
- 4) C Program to Copy a String

C PROGRAMMING FUNCTIONS

A function is a block of code that performs a specific task.

Suppose, a program related to graphics needs to create a circle and color it depending upon the radius and color from the user. You can create two functions to solve this problem:

- create a circle function
- color function

Dividing complex problem into small components makes program easy to understand and use.

Types of functions in C programming

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming

There are two types of functions in C programming:

- Standard library functions
- User defined functions

Standard library functions

The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.

These functions are defined in the header file. When you include the header file, these functions are available for use. For example:

The `printf()` is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in "`stdio.h`" header file.

There are other numerous library functions defined under "`stdio.h`", such as `scanf()`, `fprintf()`, `getchar()` etc. Once you include "`stdio.h`" in your program, all these functions are available for use.

User-defined functions

As mentioned earlier, C allow programmers to define functions. Such functions created by the user are called user-defined functions.

Depending upon the complexity and requirement of the program, you can create as many user-defined functions as you want.

How user-defined function works?

```
#include <stdio.h>
void functionName()
{
    ... ..
    ... ..
}
```

```
int main()
{
    ....
    ....

    functionName();

    ....
    ....
}
```

The execution of a C program begins from the `main()` function.

When the compiler encounters `functionName();` inside the main function, control of the program jumps to

```
void functionName()
```

And, the compiler starts executing the codes inside the user-defined function.

The control of the program jumps to statement next to `functionName();` once all the codes inside the function definition are executed.

How function works in C programming?

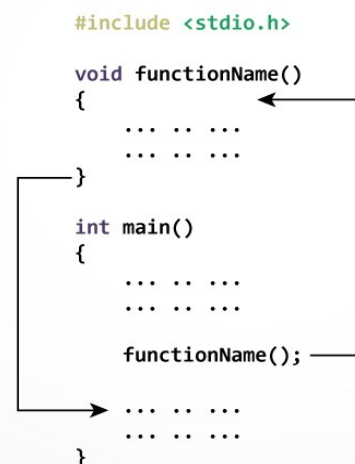
Remember, function name is an identifier and should be unique.

This is just an overview on user-defined function. Visit these pages to learn more on:

- User-defined Function in C programming
- Types of user-defined Functions

Advantages of user-defined function

1. The program will be easier to understand, maintain and debug.
2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.



A function is a block of code that performs a specific task.

C allows you to define functions according to your need. These functions are known as user-defined functions. For example:

Suppose, you need to create a circle and color it depending upon the radius and color. You can create two functions to solve this problem:

- createCircle() function
- color() function

Example: User-defined function

Here is a example to add two integers. To perform this task, a user-defined function `addNumbers()` is defined.

```
#include <stdio.h>
int addNumbers(int a, int b);    // function prototype
int main()
{
    int n1,n2,sum;
    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);
    sum = addNumbers(n1, n2);    // function call
    printf("sum = %d",sum);
    return 0;
}
int addNumbers(int a,int b)      // function definition
{
    int result;
    result = a+b;
    return result;               // return statement
}
```

Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

Syntax of function prototype

```
returnType functionName(type1 argument1, type2 argument2,...);
```

In the above example, `int addNumbers(int a, int b);` is the function prototype which provides following information to the compiler:

1. name of the function is `addNumbers()`
2. return type of the function is `int`
3. two arguments of type `int` are passed to the function

The function prototype is not needed if the user-defined function is defined before the `main()` function.

Calling a function

Control of the program is transferred to the user-defined function by calling it.

Syntax of function call

```
functionName(argument1, argument2, ...);
```

In the above example, function call is made using `addNumbers(n1,n2);` statement inside the `main()`.

Function definition

Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it.

Syntax of function definition

```
returnType functionName(type1 argument1, type2 argument2, ...)  
{  
    //body of the function  
}
```

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.


Passing arguments to a function

In programming, argument refers to the variable passed to the function. In the above example, two variables `n1` and `n2` are passed during function call.

The parameters `a` and `b` accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.

How to pass arguments to a function?

```
#include <stdio.h>  
  
int addNumbers(int a, int b);  
  
int main()  
{  
    ... ..  
  
    sum = addNumbers(n1, n2);  
    ... ..  
}  
  
int addNumbers(int a, int b)  
{  
    ... ..  
    ... ..  
}
```



The diagram shows two arrows originating from the arguments `n1` and `n2` in the function call `addNumbers(n1, n2);` within the `main()` function. One arrow points to the parameter `a` in the function definition `addNumbers(int a, int b)`, and the other arrow points to the parameter `b`.

The type of arguments passed to a function and the formal parameters must match, otherwise the compiler throws error.

If `n1` is of char type, `a` also should be of char type. If `n2` is of float type, variable `b` also should be of float type.

A function can also be called without passing an argument.

Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement.

In the above example, the value of variable `result` is returned to the variable `sum` in the `main()` function.

Syntax of return statement

```
return (expression);
```

For example,

```
return a;  
return (a+b);
```

The type of value returned from the function and the return type specified in function prototype and function definition must match.

Return statement of a Function

```
#include <stdio.h>  
  
int addNumbers(int a, int b);  
  
int main()  
{  
    ... ..  
    sum = addNumbers(n1, n2);  
    ... ..  
}  
  
int addNumbers(int a, int b)  
{  
    ... ..  
    return result;  
}
```

sum = result

For better understanding of arguments and return value from the function, [user-defined functions](#) can be categorized as:

- Function with no arguments and no return value
- Function with no arguments and a return value
- Function with arguments and no return value
- Function with arguments and a return value.

The 4 programs below check whether an integer entered by the user is a prime number or not. And, all these programs generate the same output.

The 4 programs below check whether an integer entered by the user is a prime number or not. And, all these programs generate the same output.

Example #1: No arguments passed and no return Value

```
#include <stdio.h>  
void checkPrimeNumber();  
int main()  
{  
    checkPrimeNumber(); // no argument is passed to prime()  
    return 0;  
}  
// return type of the function is void because no value is returned from the function  
void checkPrimeNumber()  
{  
    int n, i, flag=0;
```

```

printf("Enter a positive integer: ");
scanf("%d",&n);
for(i=2; i <= n/2; ++i)
{
    if(n%i == 0)
    {
        flag = 1;
    }
}
if (flag == 1)
    printf("%d is not a prime number.", n);
else
    printf("%d is a prime number.", n);

```

The `checkPrimeNumber()` function takes input from the user, checks whether it is a prime number or not and displays it on the screen.

The empty parentheses in `checkPrimeNumber()`; statement inside the `main()` function indicates that no argument is passed to the function.

The return type of the function is `void`. Hence, no value is returned from the function.

Example #2: No arguments passed but a return value

```

#include <stdio.h>
int getInteger();
int main()
{
    int n, i, flag = 0;
    // no argument is passed to the function
    // the value returned from the function is assigned to n
    n = getInteger();
    for(i=2; i<=n/2; ++i)
    {
        if(n%i==0){
            flag = 1;
            break;
        }
    }
    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);
    return 0;
}
// getInteger() function returns integer entered by the user
int getInteger()
{
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    return n;
}

```


The empty parentheses in `n = getInteger();` statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to `n`.

Here, the `getInteger()` function takes input from the user and returns it. The code to check whether a number is prime or not is inside the `main()` function.

Example #3: Argument passed but no return value

```
#include <stdio.h>
void checkPrimeAndDisplay(int n);

int main()
{
    int n;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    // n is passed to the function
    checkPrimeAndDisplay(n);

    return 0;
}

// void indicates that no value is returned from the function
void checkPrimeAndDisplay(int n)
{
    int i, flag = 0;

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0){
            flag = 1;
            break;
        }
    }
    if(flag == 1)
        printf("%d is not a prime number.",n);
    else
        printf("%d is a prime number.", n);
}
```

The integer value entered by the user is passed to `checkPrimeAndDisplay()` function.

Here, the `checkPrimeAndDisplay()` function checks whether the argument passed is a prime number or not and displays the appropriate message.

Example #4: Argument passed and a return value

```
#include <stdio.h>
int checkPrimeNumber(int n);

int main()
{
```

```

int n, flag;
printf("Enter a positive integer: ");
scanf("%d",&n);
// n is passed to the checkPrimeNumber() function
// the value returned from the function is assigned to flag variable
flag = checkPrimeNumber(n);

if(flag==1)
    printf("%d is not a prime number",n);
else
    printf("%d is a prime number",n);

return 0;
}
// integer is returned from the function
int checkPrimeNumber(int n)
{
    /* Integer value is returned from function checkPrimeNumber() */
    int i;

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
            return 1;
    }
    return 0;
}

```

The input from the user is passed to `checkPrimeNumber()` function.

The `checkPrimeNumber()` function checks whether the passed argument is prime or not. If the passed argument is a prime number, the function returns 0. If the passed argument is a non-prime number, the function returns 1. The return value is assigned to `flag` variable.

Then, the appropriate message is displayed from the `main()` function.

C Programming Pointers

Pointers are used in C program to access the memory and manipulate the address.

Address in C

Before you get into the concept of pointers, let's first get familiar with address in C.

If you have a variable `var` in your program, `&var` will give you its address in the memory, where `&` is commonly called the reference operator.

You must have seen this notation while using `scanf()` function. It was used in the function to store the user inputted value in the address of `var`.

```
scanf("%d", &var);
```

```
/* Example to demonstrate use of reference operator in C programming. */
#include <stdio.h>
int main()
{
    int var = 5;
    printf("Value: %d\n", var);
    printf("Address: %u", &var); //Notice, the ampersand(&) before var.
    return 0;
}
```

Output

```
Value: 5
Address: 2686778
```

Note: You may obtain different value of address while using this code.

In above source code, value 5 is stored in the memory location 2686778. `var` is just the name given to that location.

Pointer variables

In C, there is a special variable that stores just the address of another variable. It is called Pointer variable or, simply, a pointer.

Declaration of Pointer

```
data_type* pointer_variable_name;
```

```
int* p;
```

Above statement defines, `p` as pointer variable of type `int`.

Reference operator (&) and Dereference operator (*)

As discussed, `&` is called reference operator. It gives you the address of a variable.

Likewise, there is another operator that gets you the value from the address, it is called a dereference operator (`*`).

Below example clearly demonstrates the use of pointers, reference operator and dereference operator.

Note: The `*` sign when declaring a pointer is not a dereference operator. It is just a similar notation that creates a pointer.

Example To Demonstrate Working of Pointers

```
/* Source code to demonstrate, handling of pointers in C program */
#include <stdio.h>
int main() {
    int* pc;
    int c;
    c=22;
    printf("Address of c:%u\n",&c);
    printf("Value of c:%d\n\n",c);
    pc=&c;
    printf("Address of pointer pc:%u\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);
    c=11;
    printf("Address of pointer pc:%u\n",pc);
    printf("Content of pointer pc:%d\n\n",*pc);
    *pc=2;
    printf("Address of c:%u\n",&c);
    printf("Value of c:%d\n\n",c);
    return 0;
}
```

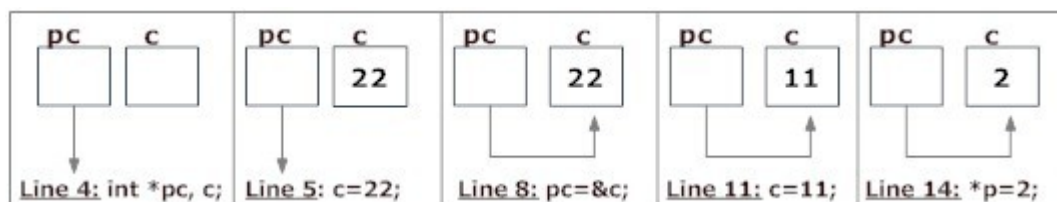
Output

Address of c: 2686784
Value of c: 22

Address of pointer pc: 2686784
Content of pointer pc: 22

Address of pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2



Explanation of program and figure

1. `int* pc;` creates a pointer `pc` and `int c;` creates a normal variable `c`.
Since `pc` and `c` are both not initialized, pointer `pc` points to either no address or a random address.
Likewise, variable `c` is assigned an address but contains a random/garbage value.
2. `c=22;` assigns 22 to the variable `c`, i.e., 22 is stored in the memory location of variable `c`.
Note that, when printing `&c` (address of `c`), we use `%u` rather than `%d` since address is usually expressed as an unsigned integer (always positive).
3. `pc=&c;` assigns the address of variable `c` to the pointer `pc`.
When printing, you see value of `pc` is the same as the address of `c` and the content of `pc` (`*pc`) is 22 as well.
4. `c=11;` assigns 11 to variable `c`.
We assign a new value to `c` to see its effect on pointer `pc`.
5. Since, pointer `pc` points to the same address as `c`, value pointed by pointer `pc` is 11 as well.
Printing the address and content of `pc` shows the updated content as 11.
6. `*pc=2;` changes the contents of the memory location pointed by pointer `pc` to 2.
Since the address of pointer `pc` is same as address of `c`, value of `c` also changes to 2.

Common mistakes when working with pointers

Suppose, you want pointer `pc` to point to the address of `c`. Then,

```
int c, *pc;
```

```
// Wrong! pc is address whereas, c is not an address.
```

```
pc = c;
```

```
// Wrong! *pc is the value pointed by address whereas, &c is an address.
```

```
*pc = &c;
```

```
// Correct! pc is an address and, &pc is also an address.
```

```
pc = &c;
```

```
// Correct! *pc is the value pointed by address and, c is also a value.
```

```
*pc = c;
```

In both cases, pointer pc is not pointing to the address of c.