Computer Problem Solving using C

GCW M.A Road
Computer problem solving is an intricate/complicated process requiring much thought, careful planning, logical precision, persistence and attention to detail.

Problem solving is the act of finding a solution to a perplexing, distressing, vexing, or unsettled question.

Program and Algorithms

- The vehicle for the computer solution to a problem is a set of explicit and unambiguous instructions expressed in a programming language.
- This set of instructions is called a program.
- A program may also be thought of as an algorithm expressed in a programming language.

An algorithm therefore corresponds to a solution to a problem that is independent of any programming language.

- An algorithm consists of a set of explicit and unambiguous finite steps which, when carried out for a given set of initial conditions, produce the corresponding output and terminate in a finite time.

To obtain computer solution to a problem we supply the program with input or data.

- The program then manipulates it according to its instructions and produces an output which represents the computer solution to the problem.

THE PROBLEM-SOLVING ASPECT

- It is widely recognized that problem-solving is a creative process which largely defies systematization and mechanization.
The plain fact of the matter is that there is no universal method for problem-solving.

Different strategies appear to work for different people.

**Problem definition phase**

Success in solving any problem is only possible after we have made the effort to come to terms with or understand the problem at hand.

We cannot hope to make useful progress in solving a problem until we fully understand what it is we are trying to solve.

This preliminary investigation may be thought of as the problem definition phase.

In other words, what we must do during this phase is work out what must be done rather than how to do it.

That is, we must try to extract from the problem statement (which is often quite imprecise and maybe even ambiguous) a set of precisely defined tasks.

—the sooner you start coding your program the longer it is going to take—

So think first and code later!

A skill that it is important to try to develop in problem-solving is the ability to view a problem from a variety of angles.

One must be able to metaphorically turn a problem upside down, inside out, sideways, backwards, forwards and so on. Once one has developed this skill it should be possible to get started on any problem.

**Computer Problem-Solving (Using computer as a problem solving tool)**
Figure 6.2 The computer problem-solving process

Figure 6.3: The Interactions between Problem-Solving Phases

Problem-Solving Phase

1. Concrete Solution (Program). Translate the algorithm (the general solution) into a programming language.

2. Test. Have the computer follow the instructions. Then manually check the results. If you find errors, analyze the program and the algorithm to determine the source of the errors, and then make corrections.

Once a program has been written, it enters a third phase: maintenance.
Maintenance Phase

1. Use. Use the program.

2. Maintain. Modify the program to meet changing requirements or to correct any errors that show up while using it.

General problem-solving strategies

Most widely known and most often used of these principles is the divide-and-conquer strategy.

- Break up a large problem into smaller units that we can handle
- Applies the concept of abstraction
- The divide-and-conquer approach can be applied over and over again until each subtask is manageable

This way of breaking down the solution to a problem has found wide application in particular with sorting, selection, and searching algorithms.

Another general problem-solving strategy that we will briefly consider is that of dynamic programming.
This method is used most often when we have to build up a solution to a problem via a sequence of intermediate steps.

A dynamic programming algorithm remembers past results and uses them to find new results.

Dynamic programming is generally used for optimization problems.

Multiple solutions exist, need to find the best one.

Requires optimal substructure and overlapping sub-problems.

This differs from Divide and Conquer, where subproblems generally need not overlap.

This method relies on the idea that a good solution to a large problem can sometimes be built up from good or optimal solutions to smaller problems.

This type of strategy is particularly relevant for many optimization problems that one frequently encounters in operations research.

The techniques of greedy search, backtracking and branch-and-bound evaluations are all variations on the basic dynamic programming idea.

Algorithm- (ăl`gərĭth′əm) or algorism (–rĭz′əm) [for Mohd. Al-Khowarizmi]

The programmer begins the programming process by analyzing the problem, breaking it into manageable pieces, and developing a general solution for each piece called an algorithm.
The solutions to the pieces are collected together to form a program that solves the original problem.

An algorithm is a written or verbal description of a logical sequence of actions applied to objects. We use algorithms every day. Recipes, instructions, and directions are all examples of algorithms that are not programs.

Algorithm any method, procedure, or set of instructions for carrying out a task by means of a precisely specified series of steps or sequence of actions, e.g. as in long division, the hierarchical sequence of steps in a typical computer program, or the steps in a manufacturing process.

**Testing Algorithm**

- After developing a general solution, the programmer tests the algorithm, walking through each step manually with paper and pencil.

- If the algorithm doesn't work, the programmer repeats the problem-solving process, analyzing the problem again and coming up with another algorithm.

- When the programmer is satisfied with the algorithm, he or she translates it into a programming language.

**Coding the Algorithm**

Translating an algorithm into a programming language is called coding the algorithm.

The products of the translation—the code for all the algorithms in the problem—are tested by collecting them into a program and running (executing) the program on the computer.
If the program fails to produce the desired results, the programmer must debug it—that is, determine what is wrong and then modify the program, or even one or more of the algorithms, to fix it.

The combination of coding and testing the algorithms is called implementation.

**Maintenance**

Once a program has been put into use, it is often necessary to modify it. Modification may involve fixing an error that is discovered during the use of the program or changing the program in response to changes in the user’s requirements.

Each time the program is modified, it is necessary to repeat the problem-solving and implementation phases for those aspects of the program that change.

This phase of the programming process is known as maintenance and actually accounts for the majority of the effort expended on most programs.

Together, the problem-solving, implementation, and maintenance phases constitute the program’s life cycle.

**Documentation**

In addition to solving the problem, implementing the algorithm, and maintaining the program, writing documentation is an important part of the programming process.

Documentation includes written explanations of the problem being solved and the organization of the solution, comments embedded within the program itself, and user manuals that describe how to use the program characteristics of a good algorithm
Precision / definiteness — the steps are precisely stated (defined).

Uniqueness — results of each step are uniquely defined and only depend on the input and the result of the preceding steps.

Finiteness — the algorithm stops after a finite number of instructions are executed.

Input — the algorithm receives input.

Output — the algorithm produces output.

Generality — the algorithm applies to a set of inputs.

Writing Pseudo-Code: Advice

Input/output must properly defined

All your variables must be properly initialized, introduced

Variables are instantiated, assigned using

All `commands' (while, if, repeat, begin, end) bold face \bf

For i = 1 to n Do

All functions in small caps Union(s, t) \sc

All constants in courier: pi \( \approx 3.14 \tt

All variables in italic: temperature \( \approx 78 \ (\it, \em

\LaTeX: Several algorithm formatting packages exist on WWW

Flowchart: What is a Flowchart?

The flowchart is a means of visually presenting the flow of control through an information processing systems, the operations performed within the system and the sequence in which they are performed.

It is a graphic representation of how a process works, showing, at a minimum, the sequence of steps.
Flowcharts are generally drawn in the early stages of formulating computer solutions.

**Guideline for drawing a flowchart:**
Flowcharts are usually drawn using some standard symbols; Some standard symbols, which are frequently required for flowcharting many computer programs are shown below,

**ASSEMBLER:**
Assembler is a computer program which is used to translate program written in Assembly Language in to machine language. The translated program is called as object program. Assembler checks each instruction for its correctness and generates diagnostic messages, if there are mistakes in the program.

- Translate mnemonic operation codes to their machine language equivalents. Assigning machine addresses to symbolic labels.
- Assembler directives (or pseudo-instructions) provide instructions to the assembler itself. They are not translated into machine instructions
  
  Eg: START,END

- The output of the assembler program is called the object code or object program.

- The object code is usually a machine code, also called a machine language, which can be understood directly by a specific type of CPU (central processing unit), such as x86 (i.e., Intel-compatible) or PowerPC. However, some compilers are designed to convert source code into an assembly language or some other another programming language. An assembly language is a human-readable notation for the machine language that a specific type of CPU uses.
An object code file can contain not only the object code, but also relocation information that the linker uses to assemble multiple object files to form an executable program. It can also contain other information, such as program symbols (names of variables and functions) and debugging (i.e., removing errors) information.

FLAT ASSEMBLER (FASM) are the example of one of assembler.

LABEL OPCODE OPERANDS ; COMMENTS
Ex: 10100101 01110001 LDA &71
01101001 00000001 ADD #&01
10000101 01110001 STA &71 Source code

Assembly process

Executable code

COMPILER
A compiler is a program that translates a program written in HLL to executable machine language. The process of transferring HLL source program in to object code is a lengthy and complex process as compared to assembling. Compliers have diagnostic capabilities and prompt the programmer with appropriate error message while compiling a HLL program. The corrections are to be incorporated in the program, whenever needed, and the program has to be recompiled. The process is repeated until the program is mistake free and translated to an object code.

JOB OF COMPILER
1. To translate HLL source program to machine codes.
2. To trace variables in the program
3. To include linkage for subroutines.
4. To allocate memory for storage of program and variables.

5. To generate error messages, if there are errors in the program. High-level languages such as C, C++ and Java compilers are employed. The compiler displays the list of errors and warnings for the statements violating the syntax rules of the language. Compilers also have the ability of linking subroutines of the program.

• Some examples of Compiler:

  o Microsoft Visual Studio
  o BlueJ
  o Quincy 2005

Source code

Compiler

Executable code

**INTERPRETER**

The basic purpose of interpreter is same as that of compiler. In compiler, the program is translated completely and directly executable version is generated. Whereas interpreter translates each instruction, executes it and then the next instruction is translated and this goes on until end of the program. In this case, object code is not stored and reused. Every time the program is executed, the interpreter translates each instruction freshly.
U-2 History of C

The C programming language is a structure oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie.

* C programming language features were derived from an earlier language called “B” (Basic Combined Programming Language – BCPL). B was implemented as an interpreter.

* Ritchie implemented it as a compiler, allowing the generation of machine code, declaration of data types and definition of data structures, and called it C.

* C language was invented for implementing UNIX operating system.

* In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and commonly known as K&R C.

* In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

* The C language is structured, middle level programming language developed by Dennis Ritchie.

* Operating system programs such as Windows, Unix, Linux are written in C language.

* C89/C90 and C99 are two standardized editions of C language.

* C has been written in assembly language.

Features of C programming language

* Reliability
*Portability: ability of a program to run in different environment.

*Flexibility: C combines the convenience and portable nature of a high-level language with the flexibility of a low-level language.

*Interactivity

*Modularity

*Efficiency and Effectiveness

a) Portability: The best feature for which C gained popularity is the portability of code. The C language programs could be run on variety of computers with a little or no change in the source code. Which means the language code can be used under various operating systems.

* b) Efficiency: The C language is efficient in two ways i) The source code is very compact ii) Memory Management through C is very efficient.

* c) Modularity: C allows separately compiled modules which can be linked together. The programs can be written in well structured manner. C is a language of functions. Various modules are written as functions.

* d) Pointer Operations: C is very powerful in pointer operations. Pointers can be set to various data types as well as to functions, structures etc. Arrays can be manipulated with the help of pointers.

* e) Flexible level: C programs can be written with the features of high level languages as well as that of low level languages. C thus fits in between the
two. C is not a strongly typed language. There are no bounds to number of array elements.

* f) Case Sensitivity: C is case sensitive. Which means the upper case and lower case characters are treated differently in variable names, function names etc.

**Character set**

Character set is a set of alphabets, letters and some special characters that are valid in C language.

**Alphabets**

Uppercase: A B C ................................... X Y Z

Lowercase: a b c ...................................... x y z

**Digits**

0 1 2 3 4 5 6 7 8 9

**Special Characters**

<table>
<thead>
<tr>
<th>,</th>
<th>&lt;</th>
<th>&gt;</th>
<th>.</th>
<th>_</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>)</td>
<td>;</td>
<td>$</td>
<td>:</td>
</tr>
<tr>
<td>%</td>
<td>[</td>
<td>]</td>
<td>#</td>
<td>?</td>
</tr>
<tr>
<td>`</td>
<td>&amp;</td>
<td>{</td>
<td>}</td>
<td>&quot;</td>
</tr>
<tr>
<td>^</td>
<td>!</td>
<td>*</td>
<td>/</td>
<td></td>
</tr>
</tbody>
</table>
Keywords

Keywords are predefined, reserved words used in programming that have special meaning. Keywords are part of the syntax and they cannot be used as an identifier. For example:

```c
int money;
```

Here, `int` is a keyword that indicates `money` is a `variable` of type integer.

As C is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in ANSI C.

**Keywords in C Language**

<table>
<thead>
<tr>
<th>auto</th>
<th>double</th>
<th>int</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>else</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>register</td>
<td>typedef</td>
</tr>
<tr>
<td>char</td>
<td>extern</td>
<td>return</td>
<td>union</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>signed</td>
<td>void</td>
</tr>
<tr>
<td>do</td>
<td>if</td>
<td>static</td>
<td>while</td>
</tr>
</tbody>
</table>
 Along with these keywords, C supports other numerous keywords depending upon the compiler.

All these keywords, their syntax and application will be discussed in their respective topics. However, if you want a brief information on these keywords without going further, visit list of all keywords in C programming.

**Identifiers**

Identifiers are the names you can give to entities such as variables, functions, structures etc.

Identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program. For example:

```
int money;

double accountBalance;
```

Here, money and accountBalance are identifiers.

Also remember, identifier names must be different from keywords. You cannot use int as an identifier because int is a keyword.

**Rules for writing an identifier**

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore. It is because identifier that starts with an underscore can conflict with system names. In such cases, compiler will complain about it. Some system names that start with underscore are _fileno, _iob, _wfopen etc.

3. There is no rule on the length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.

Good Programming Practice

You can choose any name for an identifier. However, if the programmer choose meaningful name for an identifier, it will be easy to understand and work on.

C Programming Variables and Constants

In this tutorial, you will learn about variables, rules for naming a variable, constants and different type of constants in C programming.

Variables

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:
int player\texttt{Score} = 95;

Here, \texttt{playerScore} is a variable of integer type. The variable is holding integer 95 in above program.

The value of an variable can be changed, hence the name 'variable'.

Rules for writing variable name in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with a system name and may cause error.
3. There is no rule on how long a variable can be. However, the first 31 characters of a variable are discriminated by the compiler. So, the first 31 letters of two variables in a program should be different.

In C programming, you have to declare a variable before you can use. It is a common practice in C programming to declare all the variables at the beginning of the program.

Constants/Literals

A constant is a value or an identifier whose value cannot be altered in a program. For example: 1, 2.5, "C programming is easy" etc.

As mentioned, an identifier also can be defined as a constant.

\texttt{const double PI = 3.14}
Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

**Integer constants**

A integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:

**Decimal constants:** 0, -9, 22 etc

**Octal constants:** 021, 077, 033 etc

**Hexadecimal constants:** 0x7f, 0x2a, 0x521 etc

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

**Floating-point constants**

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

- `-2.0`

- `0.0000234`
Note: E-5 = 10^{-5}

### Character constants

A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

### Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: \n is used for newline. The backslash (\) causes "escape" from the normal way the characters are interpreted by the compiler.

<table>
<thead>
<tr>
<th>Escape Sequences</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\f</td>
<td>Form feed</td>
</tr>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>\r</td>
<td>Return</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical tab</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>'</td>
<td>Single quotation mark</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>Double quotation mark</td>
</tr>
<tr>
<td>?</td>
<td>Question mark</td>
</tr>
<tr>
<td>\0</td>
<td>Null character</td>
</tr>
</tbody>
</table>

### String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

```
"good"        //string constant
"
"      "     //string constant of six white space
"x"          //string constant having single character.
"Earth is round\n" //prints string with newline
```
Keyword `enum` is used to define enumeration types. For example:

```cpp
enum color {yellow, green, black, white};
```

Here, `color` is a variable and `yellow`, `green`, `black` and `white` are the enumeration constants having value 0, 1, 2 and 3 respectively.
C Programming Data Types

In C programming, variables or memory locations should be declared before it can be used. Similarly, a function also needs to be declared before use.

Data types simply refers to the type and size of data associated with variables and functions.

Data types in C

1. Fundamental Data Types
   o Integer types
   o Floating type
   o Character type
2. Derived Data Types
   o Arrays
   o Pointers
   o Structures
   o Enumeration
This tutorial will focus on fundamental data types. To learn about derived data types, visit the corresponding tutorial.

**Integer data types**

Integers are whole numbers that can have both positive and negative values, but no decimal values. Example: 0, -5, 10

In C programming, keyword `int` is used for declaring integer variable. For example:

```c
int id;
```

Here, `id` is a variable of type integer.

You can declare multiple variable at once in C programming. For example:

```c
int id, age;
```

The size of `int` is either 2 bytes (In older PC's) or 4 bytes. If you consider an integer having size of 4 byte (equal to 32 bits), it can take $2^{32}$ distinct states as: $-2^{31}, -2^{31} + 1, ..., -2, -1, 0, 1, 2, ..., 2^{31} - 2, 2^{31} - 1$

Similarly, `int` of 2 bytes, it can take $2^{16}$ distinct states from $-2^{15}$ to $2^{15} - 1$. If you try to store larger number than $2^{31} - 1$, i.e., +2147483647 and smaller number than $-2^{31}$, i.e., -2147483648, program will not run correctly.

**Floating types**

Floating type variables can hold real numbers such as: 2.34, -9.382, 5.0 etc. You can declare a floating point variable in C by using either `float` or `double` keyword. For example:

```c
float accountBalance;
```
double bookPrice;

Here, both accountBalance and bookPrice are floating type variables.

In C, floating values can be represented in exponential form as well. For example:

```c
float normalizationFactor = 22.442e2;
```

**Difference between float and double**

The size of float (single precision float data type) is 4 bytes. And the size of double (double precision float data type) is 8 bytes. Floating point variables has a precision of 6 digits whereas the the precision of double is 14 digits.

**Character types**

Keyword char is used for declaring character type variables. For example:

```c
char test = 'h'
```

Here, test is a character variable. The value of test is 'h'.

The size of character variable is 1 byte.

**C Qualifiers**

Qualifiers alters the meaning of base data types to yield a new data type.

**Size qualifiers**
Size qualifiers alters the size of a basic type. There are two size qualifiers, long and short. For example:

```c
long double i;
```

The size of float is 8 bytes. However, when long keyword is used, that variable becomes 10 bytes.

If you know that the value of a variable will not be large, short can be used.

**Sign qualifiers**

Integers and floating point variables can hold both negative and positive values. However, if a variable needs to hold positive value only, unsigned data types are used. For example:

```c
// unsigned variables cannot hold negative value
unsigned int positiveInteger;
```

There is another qualifier signed which can hold both negative and positive only. However, it is not necessary to define variable signed since a variable is signed by default.

An integer variable of 4 bytes can hold data from $-2^{31}$ to $2^{31}-1$. However, if the variable is defined as unsigned, it can hold data from 0 to $2^{32}-1$.

It is important to note that, sign qualifiers can be applied to int and char types only.

**Constant qualifiers**

An identifier can be declared as a constant. To do so const keyword is used.
const int cost = 20;

The value of cost cannot be changed in the program.

**Volatile qualifiers**

A variable should be declared volatile whenever its value can be changed by some external sources outside the program. Keyword volatile is used for creating volatile variables.
C Programming Input Output (I/O): printf() and scanf()

This tutorial focuses on two in-build functions printf() and scanf() to perform I/O task in C programming. Also, you will learn how you write a valid program in C programming has several in-build library functions to perform input and output tasks.

Two commonly used functions for I/O (Input/Output) are printf() and scanf().

The scanf() function reads formatted input from standard input (keyboard) whereas the printf() function sends formatted output to the standard output (screen).

Example #1: C Output

```c
#include <stdio.h>  //This is needed to run printf() function.

int main()
{
    printf("C Programming");  //displays the content inside quotation
```
return 0;
}

Output

C Programming

How this program works?

- All valid C program must contain the main() function. The code execution begins from the start of main() function.
- The printf() is a library function to send formatted output to the screen. The printf() function is declared in "stdio.h" header file.
- Here, stdio.h is a header file (standard input output header file) and #include is a preprocessor directive to paste the code from the header file when necessary. When the compiler encounters printf() function and doesn't find stdio.h header file, compiler shows error.
- The return 0; statement is the "Exit status" of the program. In simple terms, program ends.

Example #2: C Integer Output

```
#include <stdio.h>

int main()
{
    int testInteger = 5;
```
printf("Number = %d", testInteger);

return 0;

}

Output

Number = 5

Inside the quotation of printf() function, there is a format string "%d" (for integer). If the format string matches the argument (testInteger in this case), it is displayed on the screen.

Example #3: C Integer Input/Output

#include <stdio.h>

int main()
{

    int testInteger;

    printf("Enter an integer: ");

    scanf("%d", &testInteger);

    printf("Number = %d", testInteger);

    return 0;
}
Output

Enter an integer: 4

Number = 4

The scanf() function reads formatted input from the keyboard. When user enters an integer, it is stored in variable testInteger. Note the ' & ' sign before testInteger; & testInteger gets the address of testInteger and the value is stored in that address.

Example #3: C Floats Input/Output

```c
#include <stdio.h>

int main()
{
    float f;

    printf("Enter a number: ");

    // %f format string is used in case of floats
    scanf("%f", &f);

    printf("Value = %f", f);
}
```
```c
return 0;
}
```

Output

Enter a number: 23.45

Value = 23.450000

The format string "%f" is used to read and display formatted in case of floats.

**Example #4: C Character I/O**

```c
#include <stdio.h>

int main()
{

    char var1;

    printf("Enter a character: ");

    scanf("%c",&var1);

    printf("You entered %c.",var1);

    return 0;

```
Format string \texttt{\%c} is used in case of character types.

\textbf{Little bit on ASCII code}

When a character is entered in the above program, the character itself is not stored. Instead a numeric value (ASCII value) is stored. And when we displayed that value using \texttt{\%c} text format, the entered character is displayed.

\textbf{Example \#5: C ASCII Code}

```c
#include <stdio.h>

int main()
{
    char var1;
    printf("Enter a character: ");
    scanf("\%c",\&var1);
}
```
// When %c text format is used, character is displayed in case of character types

printf("You entered \%c.\n", var1);

// When %d text format is used, integer is displayed in case of character types

printf("ASCII value of \%c is \%d.\n", var1, var1);

return 0;
}

Output

Enter a character: g

You entered g.

ASCII value of g is 103.

The ASCII value of character 'g' is 103. When, 'g' is entered, 103 is stored in variable var1 instead of g.

You can display a character if you know ASCII code of that character. This is shown by following example.
Example #6: C ASCII Code

```c
#include <stdio.h>

int main()
{
    int var1 = 69;
    printf("Character having ASCII value 69 is \%c.\n", var1);
    return 0;
}
```

Output

Character having ASCII value 69 is E.

Click here to learn more about the complete ASCII reference.

More on Input/Output of floats and Integers

Integer and floats can be displayed in different formats in C programming.

Example #7: I/O of Floats and Integers
#include <stdio.h>

int main()
{

    int integer = 9876;

    float decimal = 987.6543;

    // Prints the number right justified within 6 columns
    printf("4 digit integer right justified to 6 column: %6d\n", integer);

    // Tries to print number right justified to 3 digits but the number is not right adjusted because there are only 4 numbers
    printf("4 digit integer right justified to 3 column: %3d\n", integer);

    // Rounds to two digit places
printf("Floating point number rounded to 2 digits: \\
\%.2f\n",decimal);

// Rounds to 0 digit places

printf("Floating point number rounded to 0 digits: \\
%.f\n",987.6543);

// Prints the number in exponential notation(scientific notation)

printf("Floating point number in exponential form: \\
%e\n",987.6543);

return 0;
}

Output

4 digit integer right justified to 6 column:  9876
4 digit integer right justified to 3 column: 9876
Floating point number rounded to 2 digits: 987.65
Floating point number rounded to 0 digits: 988
Floating point number in exponential form: 9.876543e+02
C Programming Operators

C programming has various operators to perform tasks including arithmetic, conditional and bitwise operations. You will learn about various C operators and how to use them in this tutorial.

An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition.

C programming has wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as:

<table>
<thead>
<tr>
<th>Operators in C programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic Operators</strong></td>
</tr>
<tr>
<td><strong>Increment and Decrement Operators</strong></td>
</tr>
<tr>
<td><strong>Assignment Operators</strong></td>
</tr>
</tbody>
</table>
C Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning of Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition or unary plus</td>
</tr>
<tr>
<td>-</td>
<td>subtraction or unary minus</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>remainder after division (modulo division)</td>
</tr>
</tbody>
</table>

Example #1: Arithmetic Operators

```c
// C Program to demonstrate the working of arithmetic operators
```
```c
#include <stdio.h>

int main()
{
    int a = 9, b = 4, c;

    c = a + b;
    printf("a+b = %d \n", c);

    c = a - b;
    printf("a-b = %d \n", c);

    c = a * b;
    printf("a*b = %d \n", c);

    c = a / b;
    printf("a/b = %d \n", c);
}
```
c=a%b;

printf("Remainder when a divided by b = %d \n",c);

return 0;

}

Output

a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1

The operators +, - and * computes addition, subtraction and multiplication respectively as you might have expected.

In normal calculation, 9/4 = 2.25. However, the output is 2 in the program. It is because both variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after decimal point and shows answer 2 instead of 2.25.
The modulo operator % computes the remainder. When \( a = 9 \) is divided by \( b = 4 \), the remainder is 1. The % operator can only be used with integers.

Suppose \( a = 5.0 \), \( b = 2.0 \), \( c = 5 \) and \( d = 2 \). Then in C programming,

\[
a/b = 2.5 \quad // \text{Because both operands are floating-point variables}
\]
\[
a/d = 2.5 \quad // \text{Because one operand is floating-point variable}
\]
\[
c/b = 2.5 \quad // \text{Because one operand is floating-point variable}
\]
\[
c/d = 2 \quad // \text{Because both operands are integers}
\]

**Increment and decrement operators**

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

**Example #2: Increment and Decrement Operators**
// C Program to demonstrate the working of increment and decrement operators

#include <stdio.h>

int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);

    printf("--b = %d \n", --b);

    printf("++c = %f \n", ++c);

    printf("--d = %f \n", --d);

    return 0;
Output

```plaintext
++a = 11
--b = 99
++c = 11.500000
++d = 99.500000
```

Here, the operators `++` and `--` are used as prefix. These two operators can also be used as postfix like `a++` and `a--`. Visit this page to learn more on how increment and decrement operators work when used as postfix.

## C Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is `=`

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Same as</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=</code></td>
<td>a = b</td>
<td>a = b</td>
</tr>
<tr>
<td><code>+=</code></td>
<td>a += b</td>
<td>a = a+b</td>
</tr>
<tr>
<td><code>-=</code></td>
<td>a -= b</td>
<td>a = a-b</td>
</tr>
<tr>
<td><code>*=</code></td>
<td>a *= b</td>
<td>a = a*b</td>
</tr>
</tbody>
</table>
Example #3: Assignment Operators

// C Program to demonstrate the working of assignment operators

#include <stdio.h>

int main()
{
    int a = 5, c;

    c = a;
    printf("c = %d \n", c);

    c += a; // c = c+a
    printf("c = %d \n", c);
c -= a; // c = c - a

printf("c = %d \n", c);

c *= a; // c = c * a

printf("c = %d \n", c);

c /= a; // c = c / a

printf("c = %d \n", c);

c %= a; // c = c % a

printf("c = %d \n", c);

return 0;
}

Output

c = 5
C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning of Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal to</td>
<td>5 == 3 returns 0</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>5 &gt; 3 returns 1</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>5 &lt; 3 returns 0</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
<td>5 != 3 returns 1</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>5 &gt;= 3 returns 1</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>5 &lt;= 3 return 0</td>
</tr>
</tbody>
</table>
Example #4: Relational Operators

// C Program to demonstrate the working of arithmetic operators

#include <stdio.h>

int main()
{
    int a = 5, b = 5, c = 10;

    printf("%d == %d = %d \n", a, b, a == b);  // true
    printf("%d == %d = %d \n", a, c, a == c);  // false

    printf("%d > %d = %d \n", a, b, a > b);   //false
    printf("%d > %d = %d \n", a, c, a > c);   //false

    printf("%d < %d = %d \n", a, b, a < b);   //false
    printf("%d < %d = %d \n", a, c, a < c);   //true
printf("%d != %d = %d \n", a, b, a != b); //false
printf("%d != %d = %d \n", a, c, a != c); //true
printf("%d >= %d = %d \n", a, b, a >= b); //true
printf("%d >= %d = %d \n", a, c, a >= c); //false
printf("%d <= %d = %d \n", a, b, a <= b); //true
printf("%d <= %d = %d \n", a, c, a <= c); //true

return 0;
Output

```
5 == 5  = 1
5 == 10 = 0
5 > 5   = 0
5 > 10  = 0
5 < 5   = 0
5 < 10  = 1
5 != 5  = 0
5 != 10 = 1
5 >= 5  = 1
5 >= 10 = 0
5 <= 5  = 1
5 <= 10 = 1
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.
### Operator | Meaning of Operator | Example
--- | --- | ---
&& | Logical AND. True only if all operands are true | If \( c = 5 \) and \( d = 2 \) then, expression \((c == 5) && (d > 5)\) equals to 0.
|| | Logical OR. True only if either one operand is true | If \( c = 5 \) and \( d = 2 \) then, expression \((c == 5) || (d > 5)\) equals to 1.
! | Logical NOT. True only if the operand is 0 | If \( c = 5 \) then, expression \(! (c == 5)\) equals to 0.

### Example #5: Logical Operators

// C Program to demonstrate the working of logical operators

```c
#include <stdio.h>

int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
}
```
printf("(a = b) && (c > b) equals to %d \n", result);

result = (a = b) && (c < b);

printf("(a = b) && (c < b) equals to %d \n", result);

result = (a = b) || (c < b);

printf("(a = b) || (c < b) equals to %d \n", result);

result = (a != b) || (c < b);

printf("(a != b) || (c < b) equals to %d \n", result);

result = !(a != b);

printf("!(a == b) equals to %d \n", result);
result = !(a == b);

printf("!(a == b) equals to %d \n", result);

return 0;
}

Output

(a = b) && (c > b) equals to 1
(a = b) && (c < b) equals to 0
(a = b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0

Explanation of logical operator program

- (a = b) && (c > 5) evaluates to 1 because both operands (a = b) and (c > b)is 1 (true).
- (a = b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).
- (a = b) || (c < b) evaluates to 1 because (a = b) is 1 (true).
(a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).

! (a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).

! (a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

Bitwise Operators

In processor, mathematical operations like: addition, subtraction, addition and division are done in bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Meaning of operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Bitwise exclusive OR</td>
</tr>
<tr>
<td>~</td>
<td>Bitwise complement</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Shift left</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>Shift right</td>
</tr>
</tbody>
</table>

Visit [bitwise operator in C](#) to learn more.

Other Operators
Comma Operator

Comma operators are used to link related expressions together. For example:

```c
int a, c = 5, d;
```

The sizeof operator

The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc).

Example #6: sizeof Operator

```c
#include <stdio.h>

int main()
{
    int a, e[10];

    float b;

    double c;

    char d;

    printf("Size of int=%lu bytes\n", sizeof(a));

    printf("Size of float=%lu bytes\n", sizeof(b));

    printf("Size of double=%lu bytes\n", sizeof(c));
}
```
printf("Size of char=%lu byte\n", sizeof(d));

printf("Size of integer type array having 10 elements = %lu bytes\n", sizeof(e));

return 0;

Output

Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
Size of integer type array having 10 elements = 40 bytes

**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 at first. This expression evaluates to 1 if it's and evaluates to 0 if it's false.
- If conditionalExpression is true, expression1 is evaluated.
- If conditionalExpression is false, expression2 is evaluated.

Example #6: C conditional Operator

```c
#include <stdio.h>

int main(){
    char February;
    int days;

    printf("If this year is leap year, enter 1. If not enter any integer: ");

    scanf("%c",&February);

    // If test condition (February == '1') is true, days equal to 29.
    // If test condition (February =='1') is false, days equal to 28.
    days = (February == '1') ? 29 : 28;
```
```c
printf("Number of days in February = %d", days);

return 0;
}
```

Output

If this year is leap year, enter 1. If not enter any integer: 1

Number of days in February = 29

Other operators such as & (reference operator), * (dereference operator) and -> (member selection) operator will be discussed in C pointers.
C Programming Introduction

Examples

This page provides examples on basics of C programming language like: input/output, arithmetic operations etc. To understand these example, you should have basic knowledge on following topics:

1. Variables and Constants
2. Data Types
3. Input and Output in C programming
4. Operators

C Introduction Examples

<table>
<thead>
<tr>
<th>C Programming Introduction Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C Program to Print a Sentence</strong></td>
</tr>
<tr>
<td><strong>C Program to Print a Integer Entered by a User</strong></td>
</tr>
<tr>
<td><strong>C Program to Add Two Integers Entered by User</strong></td>
</tr>
<tr>
<td><strong>C Program to Multiply two Floating Point Numbers</strong></td>
</tr>
<tr>
<td><strong>C Program to Find ASCII Value of Character Entered by User</strong></td>
</tr>
<tr>
<td><strong>C Program to Find Quotient and Remainder of Two Integers Entered by User</strong></td>
</tr>
<tr>
<td><strong>C Program to Find Size of int, float, double and char of Your System</strong></td>
</tr>
<tr>
<td>C Program to Demonstrate the Working of Keyword long</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>C Program to Swap Two numbers Entered by User</td>
</tr>
</tbody>
</table>
C Program to Print an Integer Entered by User

In this program, an integer entered by the user is stored in a variable. Then, the stored data is displayed on the screen using printf() function.

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Variables and Constants
- C Programming Data Types
- C Programming Input Output (I/O): printf() and scanf()

Program to Print an Integer

```c
#include <stdio.h>

int main()
{
    int number;

    // printf() displays the formatted output
    printf("Enter an integer: ");
```

// scanf() reads the formatted input and stores them

    scanf("%d", &number);

// printf() displays the formatted output

    printf("You entered: %d", number);

    return 0;

}

Output

Enter a integer: 25

You entered: 25

In this program, an integer variable number is declared.

The printf() function displays "Enter an integer: " on the screen. Then, the scanf() function reads an integer data from the user and stores in variablenumber.

Finally, the value stored in the variable number is displayed on the screen using printf() function.
C Program to Add Two Integers

In this program, user is asked to enter two integers. Then, the sum of those two integers is stored in a variable and displayed on the screen.

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Data Types
- C Programming Variables and Constants
- C Programming Input Output (I/O): printf() and scanf()
- C Programming Operators

Program to Add Two Integers

```c
#include <stdio.h>

int main()
{

    int firstNumber, secondNumber, sumOfTwoNumbers;

    printf("Enter two integers: ");

    // Two integers entered by user is stored using scanf() function
```
```c
scanf("%d %d", &firstNumber, &secondNumber);

// sum of two numbers is stored in variable
sumOfTwoNumbers = firstNumber + secondNumber;

// Displays sum
printf("%d + %d = %d", firstNumber, secondNumber, sumOfTwoNumbers);

return 0;
}
```

Output

Enter two integers: 12

11

12 + 11 = 23
In this program, user is asked to enter two integers. Two integers entered by the user is stored in variables firstNumber and secondNumber respectively. This is done using scanf() function.

Then, variables firstNumber and secondNumber is added using + operator and the result is stored in sumOfTwoNumbers.

Finally, the sumOfTwoNumbers is displayed on the screen using printf() function.
C Program to Multiply two Floating Point Numbers

In this program, user is asked to enter two numbers (floating point numbers). Then, the product of those two integers is stored in a variable and displayed on the screen.

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Variables and Constants
- C Programming Data Types
- C Programming Input Output (I/O): printf() and scanf()
- C Programming Operators

Program to Multiply Two Numbers

```c
#include <stdio.h>

int main()
{

double firstNumber, secondNumber,
productOfTwoNumbers;

printf("Enter two numbers: ");
```

// Stores two floating point numbers in variable firstNumber and secondNumber respectively

scanf("%lf %lf", &firstNumber, &secondNumber);

// Performs multiplication and stores the result in variable productOfTwoNumbers

productOfTwoNumbers = firstNumber * secondNumber;

// Result up to 2 decimal point is displayed using %.2lf

printf("Product = %.2lf", productofTwoNumbers);

return 0;
}

Output

Enter two numbers: 2.4

1.12
In this program, user is asked to enter numbers. These two numbers entered by the user is stored in variable firstNumber and secondNumber respectively. This is done using `scanf()` function.

Then, the product of firstNumber and secondNumber is evaluated and the result is stored in variable `productName`

Finally, the `productName` is displayed on the screen using `printf()` function. Notice that, the result is round to second decimal place using `%.2lf` conversion character.

**C Program to Find ASCII Value of a Character**

In C programming, a character variable holds ASCII value (an integer number between 0 an 127) rather than character itself. You will learn how to find ASCII value of a character in this program.

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Data Types
- C Programming Variables and Constants
C Programming Input Output (I/O): printf() and scanf()

A character variable holds ASCII value (an integer number between 0 and 127) rather than that character itself in C programming. That value is known as ASCII value. For example, ASCII value of 'A' is 65.

What this means is that, if you assign 'A' to a character variable, 65 is stored in that variable rather than 'A' itself.

Here is a complete list of ASCII value of characters.

Program to Print ASCII Value

```c
#include <stdio.h>

int main() {

    char c;

    printf("Enter a character: ");

    // Reads character input from the user
    scanf("%c", &c);

    // %d displays the integer value of a character
    // %c displays the actual character
```

```c
printf("ASCII value of %c = %d", c, c);

return 0;
}
```

Output

```
Enter a character: G

ASCII value of G = 71
```

In this program, user is asked to enter a character which is stored in variable `c`. The ASCII value of that character is stored in variable `c` rather than that variable itself.

When `%d` format string is used, 71 (ASCII value of 'G') is displayed. When `%c` format string is used, 'G' itself is displayed.
C Program to Compute Quotient and Remainder

This program evaluates the quotient and remainder when an integer is divided by another integer.

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Data Types
- C Programming Variables and Constants
- C Programming Input Output (I/O): printf() and scanf()
- C Programming Operators

Program to Compute Quotient and Remainder

```c
#include <stdio.h>

int main(){

    int dividend, divisor, quotient, remainder;

    printf("Enter dividend: ");

    scanf("%d", &dividend);
```
printf("Enter divisor: ");

scanf("%d", &divisor);

// Computes quotient

quotient = dividend/divisor;

// Computes remainder

remainder = dividend%divisor;

printf("Quotient = %d\n",quotient);

printf("Remainder = %d",remainder);

return 0;
}
Enter dividend: 25

Enter divisor: 4

Quotient = 6

Remainder = 1

In this program, user is asked to enter two integers (dividend and divisor) which is stored in variable dividend and divisor.

Then the quotient is evaluated using division / operator and stored in variable quotient. Similarly, the remainder is evaluated using modulus % operator and stored in remainder variable.

Finally, the quotient and remainder is displayed using printf() function.

Learn more on how division / and modulus operator % operator works in C programming.

C Program to Swap Two Numbers

This page contains two different techniques to swap numbers in C programming. The first program uses temporary variable to swap numbers, whereas second program does not use any temporary variable.

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Data Types
- C Programming Operators
- C Programming Input Output (I/O): printf() and scanf()
Program to Swap Numbers Using Temporary Variable

```c
#include <stdio.h>

int main()
{
    double firstNumber, secondNumber, temporaryVariable;

    printf("Enter first number: ");
    scanf("%lf", &firstNumber);

    printf("Enter second number: ");
    scanf("%lf", &secondNumber);

    // Value of firstNumber is assigned to temporaryVariable
    temporaryVariable = firstNumber;
}```
// Value of secondNumber is assigned to firstNumber

    firstNumber = secondNumber;

// Value of temporaryVariable (which contains the initial value of firstNumber) is assigned to secondNumber

    secondNumber = temporaryVariable;

    printf("\nAfter swapping, firstNumber = %.2lf\n", firstNumber);

    printf("After swapping, secondNumber = %.2lf", secondNumber);

    return 0;
}

Output
Enter first number: 1.20

Enter second number: 2.45

After swapping, firstNumber = 2.45

After swapping, secondNumber = 1.20

In the above program, the temporaryVariable is assigned the value of firstNumber. Then, the value of firstNumber is assigned to secondNumber. Finally, the temporaryVariable (which holds the initial value of firstNumber) is assigned to secondNumber which completes the swapping process.

Here, temporaryVariable is used to hold the value of firstNumber and doesn't have any other use except that. You can also write the swapping program without using temporaryVariable.
C Program to Display Armstrong Number Between Two Intervals

To understand this example, you should have the knowledge of following C programming topics:

- C Programming if, if..else and Nested if...else Statement
- C Programming for Loop

This program asks user to enter two integers and this program will display all Armstrong numbers between these intervals. If you don't know how to check whether a number is Armstrong or not in programming then, this program may seem little bit complex. Visit this page to learn about Armstrong number and how to check it in C programming.

C program to Display Armstrong Number Between Intervals

/* Source Code to display Armstrong number between two intervals entered by user. */

#include <stdio.h>

int main()
{

```c
int n1, n2, i, temp, num, rem;

printf("Enter two numbers(intervals): ");

scanf("%d %d", &n1, &n2);

printf("Armstrong numbers between %d an %d are: ", n1, n2);

for(i=n1+1; i<n2; ++i)
{

temp=i;

num=0;

while(temp!=0)
{

rem=(temp%10);

num+=rem*rem*rem;

temp/=10;

}

if(i==num)
{
```
printf("%d ",i);

}
}

return 0;
}

Output

Enter two numbers(intervals): 100

400

Armstrong numbers between 100 and 400 are: 153 370 371

In this program, it is assumed that, the user always enters smaller number first. This program will not perform the task intended if user enters larger number first. You can add the code to swap two numbers entered by user if user enters larger number first to make this program work properly.
C Program to Display Prime Numbers Between Two Intervals

To understand this example, you should have the knowledge of following C programming topics:

- C Programming if, if..else and Nested if...else Statement
- C Programming for Loop
- C Programming break and continue Statement

This program asks user to enter two integers and this program will display all prime numbers between these intervals. If you don't know how to check whether a number is prime or not then, this program may seem little bit complex. You can visit this page to learn about prime numbers and how to check whether a number is prime or not in C programming.

Source Code to Display Prime Numbers Between two Intervals

```c
/* C program to display all prime numbers between Two interval entered by user. */

#include <stdio.h>

int main()
{

    int n1, n2, i, j, flag;

    printf("Enter two numbers(intervals): ");
```
```c
scanf("%d %d", &n1, &n2);

printf("Prime numbers between %d and %d are: ", n1, n2);

for(i=n1+1; i<n2; ++i)
{
    flag=0;

    for(j=2; j<=i/2; ++j)
    {
        if(i%j==0)
        {
            flag=1;
            break;
        }
    }

    if(flag==0)
    {
        printf("%d ",i);
    }
}
```
```c
    return 0;
}
```

Output

Enter two numbers(intervals): 20

50

Prime numbers between 20 and 50 are: 23 29 31 37 41 43 47

In this program, it is assumed that, the user always enters smaller number first. This program will not perform the task intended if user enters larger number first. You can add the code to swap two numbers entered by user if user enters larger number first to make this program work properly.

Visit this page to learn, how you can display all prime numbers between two intervals by making user-defined function.
C Program to Check Whether a Number is Prime or Not

To understand this example, you should have the knowledge of following C programming topics:

- C Programming if, if..else and Nested if...else Statement
- C Programming for Loop
- C Programming break and continue Statement

A positive integer which is only divisible by 1 and itself is known as prime number. For example: 13 is a prime number because it is only divisible by 1 and 13 but, 15 is not prime number because it is divisible by 1, 3, 5 and 15.

C program to Check Prime Number

```c
/* C program to check whether a number is prime or not. */

#include <stdio.h>

int main()
{
    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;
        break;
    }
}

if (flag==0)
    printf("%d is a prime number. ",n);
else
    printf("%d is not a prime number. ",n);

return 0;
Enter a positive integer: 29

29 is a prime number.

This program takes a positive integer from user and stores it in variable n. Then, for loop is executed which checks whether the number entered by user is perfectly divisible by i or not starting with initial value of i equals to 2 and increasing the value of i in each iteration. If the number entered by user is perfectly divisible by i then, flag is set to 1 and that number will not be a prime number but, if the number is not perfectly divisible by i until test condition i<=n/2 is true means, it is only divisible by 1 and that number itself and that number is a prime number.
C Program to Check Whether a Number is Positive or Negative

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement

This program takes a number from user and checks whether that number is either positive or negative or zero.

Source Code

```c
#include <stdio.h>

int main()

{

    float num;

    printf("Enter a number: ");

    scanf("%f",&num);

    if (num<=0)

    {
```

```c
```
This program also can be solved using nested if else statement.

#include <stdio.h>
```c
int main()
{
    float num;

    printf("Enter a number: ");

    scanf("%f", &num);

    if (num < 0) /* Checking whether num is less than 0*/
        printf("%.2f is negative.", num);
    else if (num > 0) /* Checking whether num is greater than zero*/
        printf("%.2f is positive.", num);
    else
        printf("You entered zero.");

    return 0;
}

Output 1
Enter a number: 12.3

12.30 is positive.

Output 2

Enter a number: 0

You entered zero.
C Program to Check Whether a Number is Even or Odd

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement

Numbers perfectly divisible by 2 are known even numbers and numbers which are not divisible by 2 are called odd numbers. This program takes an integer from user and checks whether that number is even or odd and displays the result.

Source Code to Check Whether a Number is Even or Odd

```c
/*C program to check whether a number entered by user is even or odd. */

#include <stdio.h>

int main(){
    int num;

    printf("Enter an integer you want to check: ");
```
```c
scanf("%d",&num);

if((num%2)==0)  /* Checking whether remainder is 0 or not. */
    printf("%d is even.",num);
else
    printf("%d is odd.",num);

return 0;
}
```

Output 1

Enter an integer you want to check: 25

25 is odd.

Output 2

Enter an integer you want to check: 12

12 is even.
In this program, user is asked to enter an integer which is stored in variable `num`. Then, the remainder is found when that number is divided by 2 and checked whether remainder is 0 or not. If remainder is 0 then, that number is even otherwise that number is odd. This task is performed using `if...else` statement in C programming and the result is displayed accordingly.

This program also can be solved using **conditional operator** `?:` which is the shorthand notation for `if...else` statement.

```c
/* C program to check whether an integer is odd or even using conditional operator */

#include <stdio.h>

int main(){

    int num;

    printf("Enter an integer you want to check: ");

    scanf("%d",&num);

    ((num%2)==0) ? printf("%d is even.",num) : printf("%d is odd.",num);

    return 0;
}
```
C Program to Count Number of Digits of an Integer

To understand this example, you should have the knowledge of following [C programming](https://en.wikipedia.org/wiki/C_programming) topics:

- [C programming while and do…while Loop](https://en.wikipedia.org/wiki/C_Programming_While_and_Do…While_Loop)

This program takes an integer from user and calculates the number of digits in that integer. For example: If user enters 2319, the output of program will be 4 because it contains 4 digits.

C Program to Find Number of Digits in a Number

```c
#include <stdio.h>

int main()
{

    int n, count = 0;

    printf("Enter an integer: ");

    scanf("%d", &n);

    while(n!=0)
    {
        count++;
    }

    printf("Number of digits is %d\n", count);

    return 0;
}
```

Enter an integer: 34523

Number of digits: 5

This program takes an integer from user and stores that number in variable n. Suppose, user entered 34523. Then, while loop is executed because n!=0 will be true in first iteration. The codes inside while loop will be executed. After first iteration, value of n will be 3452 and count will be 1. Similarly, in second iteration n will be equal to 345 and count will be equal to 2. This process goes on and after fourth iteration, n will be equal to 3 and count will be equal to 4. Then, in next iteration n will be equal to 0 and count will be equal to 5 and program will be terminated as n!=0 becomes false.
C Programming if, if..else and Nested if...else Statement

The if, if...else and nested if...else statement are used to make one-time decisions in C Programming, that is, to execute some code/s and ignore some code/s depending upon the test expression.

C if Statement

```c
if (test expression) {
    statement/s to be executed if test expression is true;
}
```

The if statement checks whether the test expression inside parenthesis () is true or not. If the test expression is true, statement/s inside the body of if statement is executed but if test is false, statement/s inside body of if is ignored.

Flowchart of if statement
Example 1: C if statement

Write a C program to print the number entered by user only if the number entered is negative.

```c
#include <stdio.h>

int main()
{
    int num;

    printf("Enter a number to check.\n");
```
```c
scanf("%d",&num);

    if(num<0) {       /* checking whether number is
                   less than 0 or not. */

        printf("Number = %d\n",num);

    }

/*If test condition is true, statement above will be
executed, otherwise it will not be executed */

        printf("The if statement in C programming is
easy.");

return 0;
}
```

Output 1

Enter a number to check.

-2

Number = -2

The if statement in C programming is easy.

When user enters -2 then, the test expression (num<0) becomes true. Hence, Number = -2 is displayed in the screen.
**C if...else statement**

The if...else statement is used if the programmer wants to execute some statement/s when the test expression is true and execute some other statement/s if the test expression is false.

**Syntax of if...else**

```
if (test expression) {
    statements to be executed if test expression is true;
}
else {
    statements to be executed if test expression is false;
}
```
Flowchart of if...else statement

![Flowchart of if...else statement](image)

Figure: Flowchart of if...else Statement

Example 2: C if...else statement

Write a C program to check whether a number entered by user is even or odd

```c
#include <stdio.h>

int main(){
    int num;
```
```c
printf("Enter a number you want to check.\n");

scanf("%d",&num);

if((num%2)==0) //checking whether
    remainder is 0 or not.

    printf("%d is even.",num);

else

    printf("%d is odd.",num);

return 0;
}
```

Output 1

Enter a number you want to check.

25

25 is odd.

Output 2

Enter a number you want to check.

2
2 is even.

**Nested if...else statement (if...elseif....else Statement)**

The nested if...else statement is used when program requires more than one test expression.

**Syntax of nested if...else statement.**

```plaintext
if (test expression1){
    statement/s to be executed if test expression1 is true;
}
else if(test expression2) {
    statement/s to be executed if test expression1 is false and 2 is true;
}
else if (test expression 3) {
    statement/s to be executed if test expression1 and 2 are false and 3 is true;
}
```
else {
    statements to be executed if all test expressions are false;
}

How nested if...else works?

The nested if...else statement has more than one test expression. If the first test expression is true, it executes the code inside the braces{ } just below it. But if the first test expression is false, it checks the second test expression. If the second test expression is true, it executes the statement/s inside the braces{ } just below it. This process continues. If all the test expression are false, code/s inside else is executed and the control of program jumps below the nested if...else

The ANSI standard specifies that 15 levels of nesting may be continued.

Example 3: C nested if else statement

Write a C program to relate two integers entered by user using = or > or < sign.

```c
#include <stdio.h>

int main() {
```
```c
int numb1, numb2;

printf("Enter two integers to check\n");

scanf("%d %d",&numb1,&numb2);

if(numb1==numb2) //checking whether two integers are equal.
    printf("Result: %d = %d",numb1,numb2);
else
    if(numb1>numb2) //checking whether numb1 is greater than numb2.
        printf("Result: %d > %d",numb1,numb2);
    else
        printf("Result: %d > %d",numb2,numb1);
return 0;
}
```

Output 1

Enter two integers to check.

5
3

Result: 5 > 3

Output 2

Enter two integers to check.

-4

-4

Result: -4 = -4
C Programming Loops

Loops cause program to execute the certain block of code repeatedly until test condition is false. Loops are used in performing repetitive task in programming. Consider these scenarios:

- You want to execute some code/s 100 times.
- You want to execute some code/s certain number of times depending upon input from user.

These types of task can be solved in programming using loops.

There are 3 types of loops in C programming:

1. for loop
2. while loop
3. do…while loop

for Loop Syntax

```c
for(initialization statement; test expression; update statement) {
    code/s to be executed;
}
```

How for loop works in C programming?

The initialization statement is executed only once at the beginning of the for loop. Then the test expression is checked by the program. If the test expression is false, for loop is terminated. But if test expression is true then the code/s inside body of for loop is executed and then update expression is updated. This process repeats until test expression is false.
This flowchart describes the working of for loop in C programming.

for loop example

Write a program to find the sum of first n natural numbers where n is entered by user. Note: 1,2,3… are called natural numbers.

```c
#include <stdio.h>

int main(){
```
```c
int n, count, sum=0;

printf("Enter the value of n. \n");

scanf("%d",&n);

for(count=1;count<=n;++count) //for loop terminates
if count>n
{
    sum+=count; /* this statement is equivalent to sum=sum+count */
}

printf("Sum=%d",sum);

return 0;
}
```

Output

Enter the value of n.

19

Sum=190
In this program, the user is asked to enter the value of n. Suppose you entered 19 then, count is initialized to 1 at first. Then, the test expression in the for loop, i.e., \((\text{count} \leq \text{n})\) becomes true. So, the code in the body of for loop is executed which makes sum to 1. Then, the expression \(++\text{count}\) is executed and again the test expression is checked, which becomes true. Again, the body of for loop is executed which makes sum to 3 and this process continues. When count is 20, the test condition becomes false and the for loop is terminated.

Note: Initial, test and update expressions are separated by semicolon(;).
C programming while and do...while Loop

Loops causes program to execute the certain block of code repeatedly until some conditions are satisfied, i.e., loops are used in performing repetitive work in programming.

Suppose you want to execute some code/s 10 times. You can perform it by writing that code/s only one time and repeat the execution 10 times using loop.

There are 3 types of loops in C programming:

1. for loop
2. while loop
3. do...while loop

Syntax of while loop

```c
while (test expression) {
    statement/s to be executed.
}
```

The while loop checks whether the test expression is true or not. If it is true, code/s inside the body of while loop is executed, that is, code/s inside the braces { } are executed. Then again the test expression is checked whether test expression is true or not. This process continues until the test expression becomes false.
Example of while loop

Write a C program to find the factorial of a number, where the number is entered by user. (Hints: factorial of \( n = 1 \times 2 \times 3 \times \ldots \times n \)

```c
/*C program to demonstrate the working of while loop*/

#include <stdio.h>

int main(){

    int number,factorial;

    printf("Enter a number.\n");

    scanf("%d",&number);

    factorial=1;

}```
```c
while (number>0){  /* while loop continues util
test condition number>0 is true */

    factorial=factorial*number;
    --number;
}

printf("Factorial=%d",factorial);

return 0;
}
```

Output

Enter a number.

5

Factorial=120

do...while loop

In C, do...while loop is very similar to while loop. Only difference between these two loops is that, in while loops, test expression is checked at first but, in do...while loop code is executed at first then the condition is checked. So, the code are executed at least once in do...while loops.

Syntax of do...while loops
do {
    some code/s;
}
while (test expression);

At first codes inside body of do is executed. Then, the test expression is checked. If it is true, code/s inside body of do are executed again and the process continues until test expression becomes false(zero).

Notice, there is semicolon in the end of while (); in do...while loop.

![Flowchart of do...while loop](image)

**Example of do...while loop**

Write a C program to add all the numbers entered by a user until user enters 0.

/*C program to demonstrate the working of do...while statement*/

#include <stdio.h>

int main()
{
    int sum=0,num;

    do /* Codes inside the body of do...while loops are at least executed once. */
    {
        printf("Enter a number\n");

        scanf("%d",&num);

        sum+=num;
    }

    while(num!=0);

    printf("sum=%d",sum);

    return 0;
}

Output
Enter a number

3

Enter a number

-2

Enter a number

0

sum=1

In this C program, user is asked a number and it is added with sum. Then, only the test condition in the do...while loop is checked. If the test condition is true, i.e., num is not equal to 0, the body of do...while loop is again executed until num equals to zero.
C Programming break and continue Statement

There are two statements built in C programming, break; and continue; to alter the normal flow of a program. Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used. The break; statement is also used in switch statement to exit

switch statement.

break Statement

In C programming, break is used in terminating the loop immediately after it is encountered. The break statement is used with conditional if statement.

Syntax of break statement

break;

The break statement can be used in terminating all three loops for, while and do...while loops.

Figure: Flowchart of break statement
The figure below explains the working of break statement in all three type of loops.

Example of break statement

Write a C program to find average of maximum of n positive numbers entered by user. But, if the input is negative, display the average(excluding the average of negative input) and end the program.

```c
/* C program to demonstrate the working of break
statement by terminating a loop, if user inputs negative
number*/

#include <stdio.h>

int main(){
```

NOTE: The break statement may also be used inside body of else statement.
```c
float num, average, sum;

int i, n;

printf("Maximum no. of inputs\n");

scanf("%d", &n);

for (i = 1; i <= n; ++i) {
    printf("Enter n%d: ", i);
    scanf("%f", &num);
    if (num < 0.0)
        break; // for loop breaks if num < 0.0
    sum = sum + num;
}

average = sum / (i - 1);

printf("Average=\%.2f", average);

return 0;
}
```

Output
Maximum no. of inputs

4

Enter n1: 1.5

Enter n2: 12.5

Enter n3: 7.2

Enter n4: -1

Average=7.07

In this program, when the user inputs number less than zero, the loop is terminated using break statement with executing the statement below it i.e., without executing sum=sum+num.

In C, break statements are also used in switch...case statement. You will study it in C switch...case statement chapter.

continue Statement

It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.

Syntax of continue Statement

```c
continue;
```

Just like break, continue is also used with conditional if statement.
For better understanding of how continue statements works in C programming. Analyze the figure below which bypasses some code/s inside loops using continue statement.

Example of continue statement

Write a C program to find the product of 4 integers entered by a user. If user enters 0 skip it.
// program to demonstrate the working of continue statement in C programming

#include <stdio.h>

int main()
{
    int i, num, product;

    for (i = 1, product = 1; i <= 4; ++i) {
        printf("Enter num%d:", i);

        scanf("%d", &num);

        if (num == 0)
            continue;  // In this program, when num equals to zero, it skips the statement product*=num and continue the loop. */

        product *= num;
    }

    printf("product=%d", product);

    return 0;
}
Output

Enter num1: 3
Enter num2: 0
Enter num3: -5
Enter num4: 2
product=-30
C Programming switch Statement

Decision making are needed when, the program encounters the situation to choose a particular statement among many statements. If a programmer has to choose one block of statement among many alternatives, nested if...else can be used but, this makes programming logic complex. This type of problem can be handled in C programming using switch statement.

Syntax of switch...case

```c
switch (n) {
    case constant1:
        code/s to be executed if n equals to constant1;
        break;
    case constant2:
        code/s to be executed if n equals to constant2;
        break;
    .
    .
    .
default:
```
code/s to be executed if \( n \) doesn't match to any cases;

The value of \( n \) is either an integer or a character in above syntax. If the value of \( n \) matches constant in case, the relevant codes are executed and control moves out of the switch statement. If the \( n \) doesn’t matches any of the constant in case, then the default codes are executed and control moves out of switch statement.

Example of switch...case statement

Write a program that asks user an arithmetic operator (‘+’, ‘-’, ‘*’ or ‘/’) and two operands and perform the corresponding calculation on the operands.
/* C program to demonstrate the working of switch...case statement */

/* C Program to create a simple calculator for addition, subtraction,
 multiplication and division */

#include <stdio.h>

int main() {

    char o;

    float num1, num2;

    printf("Select an operator either + or - or * or /
\n");

    scanf("%c", &o);

    printf("Enter two operands: ");

    scanf("%f%f", &num1, &num2);

    switch(o) {

        case '+':


printf("%.1f + %.1f = %.1f", num1, num2, num1+num2);

break;

case '-':

    printf("%.1f - %.1f = %.1f", num1, num2, num1-num2);

    break;

case '*':

    printf("%.1f * %.1f = %.1f", num1, num2, num1*num2);

    break;

case '/':

    printf("%.1f / %.1f = %.1f", num1, num2, num1/num2);

    break;

default:

    /* If operator is other than +, -, * or /,
    error message is shown */

    printf("Error! operator is not correct");
Output

Enter operator either + or - or * or /

* 

Enter two operands: 2.3

4.5

2.3 * 4.5 = 10.3

The break statement at the end of each case cause switch statement to exit. If break statement is not used, all statements below that case statement are also executed.
C Programming goto Statement

In C programming, goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

Syntax of goto statement

goto label;

.............

.............

.............

label:

statement;

In this syntax, label is an identifier. When, the control of program reaches to goto statement, the control of the program will jump to the label: and executes the code below it.
Example of goto statement

/* C program to demonstrate the working of goto statement. */

/* This program calculates the average of numbers entered by user. */

/* If user enters negative number, it ignores that number and calculates the average of number entered before it. */

#include <stdio.h>

int main(){
    float num, average, sum;
    int i, n;

    printf("Maximum no. of inputs: ");

    scanf("%d", &n);

    for(i=1; i<=n; ++i){
        printf("Enter n%d: ", i);
    }
}
```c
scanf("%f", &num);

if (num < 0.0)
    goto jump; /* control of the program moves to label jump */

    sum = sum + num;
}

jump:

    average = sum / (i - 1);

    printf("Average: %.2f", average);

return 0;
}
```

**Output**

Maximum no. of inputs: 4

Enter n1: 1.5

Enter n2: 12.5

Enter n3: 7.2
Enter n4: -1

Average: 7.07

Though goto statement is included in ANSI standard of C, use of goto statement should be reduced as much as possible in a program.

Reasons to avoid goto statement

Though, using goto statement give power to jump to any part of program, using goto statement makes the logic of the program complex and tangled. In modern programming, goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C program with the use of break and continue statements. In fact, any program in C programming can be perfectly written without the use of goto statement. All programmer should try to avoid goto statement as possible as they can.
C Programming Decision Making and Loops Examples

This page contains examples and source code on decision making in C programming (to choose a particular statement among many statements) and loops (to perform repeated task). To understand all the examples on this page, you should have knowledge of following topics:

1. if...else Statement
2. for Loop
3. while Loop
4. break and Continue Statement
5. switch...case

Decision Making and Loop Examples

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement

Numbers perfectly divisible by 2 are known even numbers and numbers which are not divisible by 2 are called odd numbers. This program takes an integer from user and checks whether that number is even or odd and displays the result.

Source Code to Check Whether a Number is Even or Odd
/* C program to check whether a number entered by user is even or odd. */

#include <stdio.h>

int main(){
    int num;

    printf("Enter an integer you want to check: ");

    scanf("%d", &num);

    if((num%2)==0) /* Checking whether remainder is 0 or not. */
        printf("%d is even.", num);
    else
        printf("%d is odd.", num);

    return 0;
}

Output 1
Enter an integer you want to check: 25
25 is odd.

Enter an integer you want to check: 12
12 is even.

In this program, user is asked to enter an integer which is stored in variable num. Then, the remainder is found when that number is divided by 2 and checked whether remainder is 0 or not. If remainder is 0 then, that number is even otherwise that number is odd. This task is performed using if...else statement in C programming and the result is displayed accordingly.

This program also can be solved using conditional operator[ ?: ] which is the shorthand notation for if...else statement.

/* C program to check whether an integer is odd or even using conditional operator */

#include <stdio.h>
```c
int main()
{
    int num;

    printf("Enter an integer you want to check: ");

    scanf("%d", &num);

    if((num % 2) == 0) printf("%d is even.", num);
    else printf("%d is odd.", num);

    return 0;
}
```
C Program to Check Vowel or Consonant

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement

Alphabets a, e, i, o and u are known as vowels and all alphabets except these characters are known as consonants. This program asks user to enter a character and checks whether that character is vowel or not.

Source Code to Check Whether a Character is Vowel or consonant

```c
#include <stdio.h>

int main(){
    char c;

    printf("Enter an alphabet: ");

    scanf("%c",&c);

    if(c=='a'||c=='A'||c=='e'||c=='E'||c=='i'||c=='I'||c=='o'||c=='O'||c=='u'||c=='U')
```
printf("%c is a vowel.",c);

else

    printf("%c is a consonant.",c);

return 0;
}

Output 1

Enter an alphabet: i

i is a vowel.

Output 2

Enter an alphabet: G

G is a consonant.

In this program, user is asked to enter a character which is stored in variable c. Then, this character is checked, whether it is any one of these ten characters a, A, e, E, i, I, o, O, u and U using logical OR operator ||. If that character is any one of these ten characters, that alphabet is a vowel if not that alphabet is a consonant.

This program also can be solved using conditional operator which is shorthand notation for if else statement.
/* C program to check whether a character is vowel or consonant using conditional operator */

#include <stdio.h>

int main(){

    char c;

    printf("Enter an alphabet: ");

    scanf("%c",&c);

    (c=='a'||c=='A'||c=='e'||c=='E'||c=='i'||c=='I'||c=='o'||c=='O'||c=='u'||c=='U') ? printf("%c is a vowel.",c) : printf("%c is a consonant.",c);

    return 0;
}
C Program to Find the Largest Number Among Three Numbers

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement

In this program user is asked to enter three numbers and this program will find the largest number among three numbers entered by user. This program can be solved in more than one way.

Source Code 1

```c
/* C program to find largest number using if statement only */

#include <stdio.h>

int main(){

    float a, b, c;

    printf("Enter three numbers: ");

    scanf("%f %f %f", &a, &b, &c);

    return 0;
}
```
if (a >= b && a >= c)
    printf("Largest number = %.2f", a);
if (b >= a && b >= c)
    printf("Largest number = %.2f", b);
if (c >= a && c >= b)
    printf("Largest number = %.2f", c);
return 0;
}

Source Code 2

/* C program to find largest number using if...else statement */

#include <stdio.h>

int main(){
    float a, b, c;
    printf("Enter three numbers: ");
```c
scanf("%f %f %f", &a, &b, &c);

if (a>=b)
{
    if(a>=c)
        printf("Largest number = %.2f",a);
    else
        printf("Largest number = %.2f",c);
}
else
{
    if(b>=c)
        printf("Largest number = %.2f",b);
    else
        printf("Largest number = %.2f",c);
}
return 0;
```
Source Code 3

/* C Program to find largest number using nested if...else statement */

#include <stdio.h>

int main()
{
    float a, b, c;
    printf("Enter three numbers: ");
    scanf("%f %f %f", &a, &b, &c);
    if(a>=b && a>=c)
        printf("Largest number = %.2f", a);
    else if(b>=a && b>=c)
        printf("Largest number = %.2f", b);
    else
        printf("Largest number = %.2f", c);
}
Though the technique to solve this problem is different in these three examples, output of all these program is same.

C Program to Find Factorial of a Number

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement
- C Programming for Loop

For any positive number \( n \), its factorial is given by:
factorial = 1*2*3*4....n

If a number is negative, factorial does not exist and factorial of 0 is 1.

This program takes an integer from a user. If user enters negative integer, this program will display error message and if user enters non-negative integer, this program will display the factorial of that number.

**Source Code to Find Factorial of a Number**

```c
/* C program to display factorial of an integer if user enters non-negative integer. */

#include <stdio.h>

int main()
{
    int n, count;

    unsigned long long int factorial=1;

    printf("Enter an integer: ");

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

if ( n < 0 )
{
    printf("Error!!! Factorial of negative number doesn't exist.");
}
else
{
    for (count=1; count<=n; ++count)  /* for loop terminates if count>n */
    {
        factorial*=count;           /* factorial=factorial*count */
        factorial=factorial*count;  /* factorial=factorial*count */
    }
    printf("Factorial = %lu",factorial);
}
return 0;
Error!!! Factorial of negative number doesn't exist.

Output 2

Enter an integer: 10

Factorial = 3628800

Here the type of factorial variable is declared as: unsigned long long. It is because, the factorial is always positive, so unsigned keyword is used and the factorial of a number can be pretty large. For example: the factorial of 10 is 3628800 thus, long \texttt{long keyword} is used.
C Program to Reverse a Number

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C programming while and do...while Loop

This program takes an integer number from user and reverses that number.

C Program to Reverse an Integer

```c
#include <stdio.h>

int main()
{
    int n, reverse=0, rem;

    printf("Enter an integer: ");

    scanf("%d", &n);

    while(n!=0)
    {
        rem=n%10;
        reverse=reverse*10+rem;
        n/=10;
    }
}
```
printf("Reversed Number = %d",reverse);

return 0;

Output

Enter an integer: 2345

Reversed Number = 5432
C Program to Find LCM of two Numbers

To understand this example, you should have the knowledge of following C programming topics:

- C Programming Operators
- C Programming if, if..else and Nested if...else Statement
- C programming while and do...while Loop

LCM of two integers a and b is the lowest positive integer this is perfectly divisible by both a and b. Visit this page to learn more about LCM.

Source Code to Compute LCM

```c
/* C program to find LCM of two positive integers entered by user */

#include <stdio.h>

int main()
{
    int num1, num2, max;

    printf("Enter two positive integers: ");
```
In this program, user is asked to enter two positive integers which will be stored in variable num1 and num2 respectively and largest of two integers is assigned to variable max. Then, while loop is executed and in each iteration it is checked whether max is perfectly divisible by two numbers entered by user or not. If max is not perfectly divisible, max is increased by
1 and this process goes not until max is perfectly divisible by both numbers. The test condition of while loop in above program is always true so, the loop is terminated using break statement.

The LCM of two numbers also can be found using following formula:

\[
\text{LCM} = \frac{\text{num1} \times \text{num2}}{\text{GCD}}
\]

Visit this page to learn different methods for finding GCD of two numbers.

```c
#include<stdio.h>

int main()
{

    int n1,n2,temp1,temp2;

    printf("Enter two positive integers: ");

    scanf("%d %d",&n1,&n2);

    temp1=n1;

    temp2=n2;

    while(temp1!=temp2)
    {
```
if(temp1>temp2)

    temp1-=temp2;

else

    temp2-=temp1;

}

printf("LCM of two numbers %d and %d is %d", n1, n2, (n1*n2)/temp1);

return 0;
}

The output of these two programs is same.

Output

Enter two positive numbers: 15

9

LCM of two numbers 15 and 9 is 45
C Program to Make a Simple Calculator to Add, Subtract, Multiply or Divide Using switch...case

To understand this example, you should have the knowledge of following C programming topics:

- C Programming switch Statement
- C Programming break and continue Statement

This program takes an arithmetic operator (+, -, *, /) and two operands from an user and performs the operation on those two operands depending upon the operator entered by user.

Source Code to Make Simple Calculator in C programming

/* Source code to create a simple calculator for addition, subtraction, multiplication and division using switch...case statement in C programming. */

#include <stdio.h>

int main()
{ 
    char o;

    float num1,num2;

    printf("Enter operator either + or - or * or divide : ");

    scanf("%c",&o);

    printf("Enter two operands: ");

    scanf("%f%f",&num1,&num2);

    switch(o) {

    case '+':

        printf("%.1f + %.1f = %.1f",num1, num2, num1+num2);

        break;

    case '-':

        printf("%.1f - %.1f = %.1f",num1, num2, num1-num2);

        break;

    case '*':

    }
printf("%.1f * %.1f = %.1f",num1, num2, num1*num2);

    break;

case '/':

    printf("%.1f / %.1f = %.1f",num1, num2, num1/num2);

    break;

default:

    /* If operator is other than +, -, * or /, error message is shown */

    printf("Error! operator is not correct");

    break;

}
Enter two operands: 3.4

8.4

3.4 - 8.4 = -5.0

This program takes an operator and two operands from user. The operator is stored in variable operator and two operands are stored in num1 and num2 respectively. Then, switch...case statement is used for checking the operator entered by user. If user enters + then, statements for case: '+', is executed and program is terminated. If user enters - then, statements for case: '-', is executed and program is terminated. This program works similarly for * and / operator. But, if the operator doesn't matches any of the four character [ +, -, *, and / ], default statement is executed which displays error message.
#include <stdio.h>

int main()
{

    int n,count=0;

    printf("Enter an integer: ");

    scanf("%d", &n);

    while(n!=0)
Enter an integer: 34523

Number of digits: 5

This program takes an integer from user and stores that number in variable n. Suppose, user entered 34523. Then, while loop is executed because n!=0 will be true in first iteration. The codes inside while loop will be executed. After first iteration, value of n will be 3452 and count will be 1. Similarly, in second iteration n will be equal to 345 and count will be equal to 2. This process goes on and after fourth iteration, n will be equal to 3 and count will be equal to 4. Then, in next iteration n will be equal to 0 and count will be equal to 5 and program will be terminated as n!=0 becomes false.
Control Instructions in C

- the ‘Control Instructions’ enable us to specify the order in which the various instructions in a program are to be executed by the computer.
- In other words the control instructions determine the ‘flow of control’ in a program.
- There are four types of control instructions in C. They are:

  (a) Sequence Control Instruction

  (b) Selection or Decision Control Instruction

  (c) Repetition or Loop Control Instruction

  (d) Case Control Instruction

- The Sequence control instruction ensures that the instructions are executed in the same order in which they appear in the program.
- Decision and Case control instructions allow the computer to take a decision as to which instruction is to be executed next.
- The Loop control instruction helps computer to execute a group of statements repeatedly.
- C has three major decision making instructions—the if statement, the if-else statement, and the switch statement. A fourth, somewhat less important structure is the one that uses conditional operators.
C Programming if, if...else and Nested if...else Statement

Decision making is used to specify the order in which statements are executed. In this tutorial, you will learn to put decision making in a C program using if, if..else and nested if...else statement.

C if statement

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

The `if` statement evaluates the test expression inside parenthesis.

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

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

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out relational and logical operators.

Flowchart of if statement
Example #1: C if statement

// Program to display a number if user enters negative number
// If user enters positive number, that number won't be displayed

#include <stdio.h>
int main()
{
    int number;

printf("Enter an integer: ");
scanf("%d", &number);

// Test expression is true if number is less than 0
if (number < 0)
{
    printf("You entered %d.\n", number);
}

printf("The if statement is easy. ");

return 0;
}
When user enters 5, the test expression (number < 0) becomes false and the statement inside the body of if is skipped.

**Lab Assignment**

**Example 2.1:** While purchasing certain items, a discount of 10% is offered if the quantity purchased is more than 1000. If quantity and price per item are input through the keyboard, write a program to calculate the total expenses.

**C if…else statement**

The if…else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0).

**Syntax of if…else**

```c
if (testExpression) {
    // codes inside the body of if
}
else {
    // codes inside the body of else
}
```
Lab Assignment

Example 2.3: In a company an employee is paid as under:

If his basic salary is less than Rs. 1500, then HRA = 10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary. If the employee's salary is input through the keyboard write a program to find his gross salary.

Nested if-elses
It is perfectly all right if we write an entire if-else construct within either the body of the if statement or the body of an else statement. This is called ‘nesting’of if's. This is shown in the following program.

/* A quick demo of nested if-else */
main( )
{
    int i ;
    printf ( "Enter either 1 or 2 " ) ;
    scanf ( "%d", &i ) ;
    if ( i == 1 )
        printf ( "You would go to heaven !" ) ;
    else
        {
            if ( i == 2 )
                printf ( "Hell was created with you in mind" ) ;
            else
                printf ( "How about mother earth !" ) ;
        }
}
**Forms of if**
The if statement can take any of the following forms:

(a) if ( condition )
do this ;

(b) if ( condition )
{
do this ;
   and this ;
}

(c) if ( condition )
do this ;
else
do this ;

(d) if ( condition )
{
do this ;
   and this ;
} else
{
do this ;
   and this ;
}

(e) if ( condition )
do this ;
else
{
   if ( condition )
do this ;
   else
   {
do this ;
      and this ;
   }
}

(f) if ( condition )
{
   if ( condition )
do this ;
   else
   {
do this ;
      and this ;
   }
} else
do this ;
Example: The marks obtained by a student in 5 different subjects are input through the keyboard. The student gets a division as per the following rules:
- Percentage above or equal to 60 - First division
- Percentage between 50 and 59 - Second division
- Percentage between 40 and 49 - Third division
- Percentage less than 40 - Fail

Write a program to calculate the division obtained by the student.

```c
main()
{
    int m1, m2, m3, m4, m5, per ;
    printf ("Enter marks in five subjects ");
    scanf ("%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5);
    per = (m1 + m2 + m3 + m4 + m5) / 5 ;
    if ( per >= 60 )
        printf("First division");
    if ( ( per >= 50 ) && ( per < 60 ) )
        printf("Second division");
    if ( ( per >= 40 ) && ( per < 50 ) )
        printf("Third division");
    if ( per < 40 )
        printf("Fail");
}
```
C Programming for Loop

The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied. This repetitive operation is done through a loop control instruction.

Loops are used in programming to repeat a specific block of code. You will learn how 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:

1. for loop
2. while loop
3. do...while loop

for Loop

The for allows us to specify three things about a loop in a single line:

(a) Setting a loop counter to an initial value

(b) Testing the loop counter to determine whether its value has reached the number of repetitions desired.
c. Increasing the value of loop counter each time the program segment within the loop has been executed.

The syntax of a for loop is:

```java
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 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.

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out [relational](#) and [logical operators](#).
/* Calculation of simple interest for 3 sets of p, n and r */

main ( )
{
  int p, n, count ;
  float r, si ;
  for ( count = 1 ; count <= 3 ; count = count + 1 )
  {
    printf ( "Enter values of p, n, and r " ) ;
    scanf ( "%d %d %f", &p, &n, &r ) ;
    si = p * n * r / 100 ;
    printf ( "Simple Interest = Rs.%f\n", si ) ;
  }
}

Nesting of Loops:

The way if statements can be nested, similarly whiles and fors can also be nested

/* Demonstration of nested loops */
main()
{
    int r, c, sum;
    for (r = 1; r <= 3; r++) /* outer loop */
    {
        for (c = 1; c <= 2; c++) /* inner loop */
        {
            sum = r + c;
            printf("r = %d c = %d sum = %d\n", r, c, sum);
        }
    }
}

When you run this program you will get the following output:
1 1 2
1 2 3
2 1 3
2 2 4
3 1 4
3 2 5

Here, for each value of r the inner loop is cycled through twice, with the variable c taking values from 1 to 2. The inner loop terminates when the value of c exceeds 2, and the outer loop terminates when the value of r exceeds 3.
C programming while and do...while Loop

Loops are used in programming to repeat a specific block of code. After reading this tutorial, you will learn how to create a while and do...while 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:

1. **for loop**
2. **while loop**
3. **do...while loop**

### while loop

The syntax of a while loop is:

```c
while (testExpression) {
  //codes
}
```

### How while loop works?

The while loop evaluates the test expression.

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

The general form of while is as shown below:
initialise loop counter;
while ( test loop counter using a condition )
{ do this;
and this;
increment loop counter ; }
main()
{
    int i = 1;
    while ( i <= 10 )
    {
        printf ( "%d\n", i );
        i = i + 1;
    }
}

It is not necessary that a loop counter must only be an int. It can even be a float.

What will be the output of the following program?

we have carelessly given a ; after the while.
Flowchart of while loop

Example #1: while loop

```c
// Program to find factorial of a number
// For a positive integer n, factorial = 1*2*3...n

#include <stdio.h>
int main()
{
    int number;
    long long factorial;

    printf("Enter an integer: ");
    scanf("%d", &number);

    factorial = 1;
```
// loop terminates when number is less than or equal to 0

while (number > 0)
{
    factorial *= number; // factorial = factorial*number;
    --number;
}

printf("Factorial= %lld", factorial);

return 0;

Output

Enter an integer: 5

Factorial = 120

To learn more on test expression (when test expression is evaluated to nonzero (true) and 0 (false)), check out relational and logical operators.

/* Calculation of simple interest for 3 sets of p, n and r */
main()
{
    int p, n, count;
    float r, si;

    count = 1;

    while (count <= 3)
    {
        printf("Enter values of p, n and r ");

        // Calculation code here
    }
do...while loop

The do-while loop looks like this:

do
{
    this;
    and this;
    and this;
    and this;
}
while ( this condition is true);

◇ The difference between the working of while and do-while loops. This difference is the place where the condition is tested. The while tests the condition before executing any of the statements within the while loop. As against this, the do-while tests the condition after having executed the statements within the loop. This means that do-while would execute its statements at least once, even if the condition fails for the first time. The while, on the other hand will not execute its statements if the condition fails for the first time.
main()
{
    while ( 4 < 1 )
        printf ( "Hello there \n" ) ;
}

main()
{
    do
        {
            printf ( "Hello there \n" ) ;
        }
    while ( 4 < 1 ) ;
}
C Programming break and continue Statement

In this tutorial, you will learn how to use break and continue statements to alter the program flow in loops.

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

break Statement

The break statement terminates the loop immediately when it is encountered. The break statement is used with decision making statement such as if...else.

Syntax of break statement

```c
break;
```

Flowchart of break statement
How break statement works?
Example #1: break statement

// Program to calculate the sum of maximum of 10 numbers
// Calculates sum until user enters positive number

#include <stdio.h>

int main()
{
    int i;
    double number, sum = 0.0;
```c
    for(i=1; i <= 10; ++i)
    {
        printf("Enter a n%d: ",i);
        scanf("%lf", &number);

        // If user enters negative number, loop is terminated
        if(number < 0.0)
        {
            break;
        }

        sum += number; // sum = sum + number;
    }

    printf("Sum = %.2lf",sum);

    return 0;
}

Output

Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
This program calculates the sum of maximum of 10 numbers. It’s because, when the user enters negative number, the break statement is executed and loop is terminated.

In C programming, break statement is also used with switch...case statement.

Assignment:

Write a program to determine whether a number is prime or not. A prime number is one, which is divisible only by 1 or itself.

All we have to do to test whether a number is prime or not, is to divide it successively by all numbers from 2 to one less than itself. If remainder of any of these divisions is zero, the number is not a prime. If no division yields a zero then the number is a prime number. Following program implements this logic.

```c
main( )
{
int num, i ;
printf ( "Enter a number " ) ;
scanf ( "%d", &num ) ;
i = 2 ;
while ( i <= num - 1 )
{
if ( num % i == 0 )
{
printf ( "Not a prime number" ) ;
blood ;
}
i++ ;
}
if ( i == num )
printf ( "Prime number" ) ;
```
The continue Statement:

The continue statement jumps to one line before the break statement. i.e. at the end of the body of for, do or while loop.

consider the following program.

main()
{
  int i, j;
  for ( i = 1; i <= 2 ; i++ )
  {
    for ( j = 1 ; j <= 2 ; j++ )
  }
{ 
    if (i == j)
    continue;
    printf ("\n%d %d\n", i, j);
}

1. The output of the above program would be...
2. 12
3. 21

C Programming switch...case Statement

The control statement that allows us to make a decision from the number of choices is called a switch, or more correctly a switch-case-default

Note that when the value of i equals that of j, the continue statement takes the control to the for loop (inner) bypassing rest of the statements pending execution in the for loop (inner).

The nested if...else statement allows you to execute a block code among many alternatives. If you are checking on the value of a single variable in nested if...else statement, 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.

### Syntax of switch...case

```java
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 associate 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.

```
main() {
    int i = 2;
    switch (i) {
        case 1: printf ("I am in case 1
" );
        break;
        case 2: printf ("I am in case 2
" );
        break;
        case 3: printf ("I am in case 3
" );
        break;
        default: printf ("I am in default
" );
    }
    The output of this program would be:
    I am in case 2
}
```

### switch Versus if-else Ladder

There are some things that you simply cannot do with a switch. These are:

- A float expression cannot be tested using a switch
- Cases can never have variable expressions (for example it is wrong to say case a +3:)
- Multiple cases cannot use same expressions.
C Programming goto Statement

In this tutorial, you will learn how to create a goto statement in C programming. Also, you will learn when to use a goto statement and when not to use it.

The goto statement is used to alter the normal sequence of a C program.

**Syntax of goto statement**

```
goto label;

... ... ...

... ... ...

... ... ...

label:

statement;
```

The label is an identifier. When goto statement is encountered, control of the program jumps to label: and starts executing the code.
Example: goto Statement

// Program to calculate the sum and average of maximum of 5 numbers
// If user enters negative number, the sum and average of previously entered positive number is displayed

#include <stdio.h>

int main()
{
    const int maxInput = 5;
    int i;
    double number, average, sum=0.0;

    for(i=1; i<=maxInput; ++i)
    {
        printf("%d. Enter a number: ", i);
        scanf("%lf", &number);

        // If user enters negative number, flow of program moves to label jump
        if(number < 0.0)
            goto jump;
    }
C Programming Functions

In this tutorial, you will be introduced to functions (both user-defined and standard library functions) in C programming. Also, you will learn why functions are used in programming.
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 in-built 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.

Visit this page to learn more about standard library functions in C programming.

**User-defined functions**

As mentioned earlier, C language allows programmer 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?**

```c
#include <stdio.h>

void 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

```c
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.

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

**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.

C Programming User-defined functions

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

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

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:

- createCircle() function
- color() function

Example: User-defined function

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

```c
#include <stdio.h>

int addNumbers(int a, int b); // function prototype
```
```c
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 `add()`
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 from inside the main().

**Function definition**

Function definition contains the block of code to perform a specific task.

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

If \( n_1 \) is of char type, \( a \) also should be of char type. If \( n_2 \) 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

```c
return (expression);
```

For example,

```c
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.

Types of User-defined Functions in C Programming

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 checks 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: ");
```c
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 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.

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

**Example #2: No arguments passed but a return value**

```c
#include <stdio.h>
```
```c
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);
```c
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**

```c
#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 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;
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.

**Which approach is better?**

Well, it depends on the problem you are trying to solve. In case of this problem, the last approach is better.

A function should perform a specific task. The checkPrimeNumber() function doesn't take input from the user nor it displays the appropriate message. It only checks whether a number is prime or not, which makes code modular, easy to understand and debug.
C Programming Recursion

A function that calls itself is known as recursive function. And, this technique is known as recursion.

How recursion works?

```c
void recurse()
{
    ...
    ...
    recurse();
    ...
    ...
}

int main()
{
    ...
    ...
    recurse();
    ...
    ...
}
```
The recursion continues until some condition is met to prevent it. To prevent infinite recursion, \texttt{if...else statement} (or similar approach) can be used where one branch makes the recursive call and other doesn't.

**Example: Sum of Natural Numbers Using Recursion**

\begin{verbatim}
#include <stdio.h>

int sum(int n);

int main()
{
    ... ...
    sum(n);
    ... ...
}
\end{verbatim}
```c
int number, result;

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

result = sum(number);

printf("sum=%d", result);
```

```c
int sum(int n)
{
    if (n!=0)
        return n + sum(n-1); // sum() function calls itself
    else
        return n;
}
```

Output

```
Enter a positive integer:
3
```
Initially, the `sum()` is called from the `main()` function with number passed as an argument.

Suppose, the value of \( n \) is 3 initially. During next function call, 2 is passed to the `sum()` function. In next function call, 1 is passed to the function. This process continues until \( n \) is equal to 0.

When \( n \) is equal to 0, there is no recursive call and the sum of integers is returned to the `main()` function.
Advantages and Disadvantages of Recursion

Recursion makes program elegant and cleaner. All algorithms can be defined recursively which makes it easier to visualize and prove.

If the speed of the program is vital then, you should avoid using recursion. Recursions use more memory and are generally slow.
The integer entered by the user is stored in variable \( n \). Then the while loop is iterated until the test expression \( n \neq 0 \) is evaluated to 0 (false).

- After first iteration, the value of \( n \) will be 345 and the count is incremented to 1.
- After second iteration, the value of \( n \) will be 34 and the count is incremented to 2.
- After third iteration, the value of \( n \) will be 3 and the count is incremented to 3.
- After fourth iteration, the value of \( n \) will be 0 and the count is incremented to 4.
- Then the test expression is evaluated to false and the loop terminates.

## C Programming Arrays

In C programming, one of the frequently arising problem is to handle similar types of data. For example: If the user want to store marks of 100 students. This can be done by creating 100 variable individually but, this process is rather tedious and impracticable. These type of problem can be handled in C programming using arrays.

An array is a sequence of data item of homogeneous value(same type).

Arrays are of two types:

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

### Declaration of one-dimensional array

```c
data_type array_name[array_size];
```
For example:

```c
int age[5];
```

Here, the name of array is `age`. The size of array is 5, i.e., there are 5 items (elements) of array `age`. All elements in an array are of the same type (int, in this case).

**Array elements**

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:

```c
int age[5];
```

Note that, the first element is numbered 0 and so on.

Here, the size of array `age` is 5 times the size of int because there are 5 elements.

Suppose, the starting address of `age[0]` is 2120d and the size of int be 4 bytes. Then, the next address (address of `a[1]`) will be 2124d, address of `a[2]` will be 2128d and so on.

**Initialization of one-dimensional array:**

Arrays can be initialized at declaration time in this source code as:
int age[5]={2,4,34,3,4};

It is not necessary to define the size of arrays during initialization.

int age[]={2,4,34,3,4};

In this case, the compiler determines the size of array by calculating the number of elements of an array.

Accessing array elements

In C programming, arrays can be accessed and treated like variables in C.

For example:

```c
scanf("%d",&age[2]);
/* statement to insert value in the third element of array age[]. */
```

```c
scanf("%d",&age[i]);
/* Statement to insert value in (i+1)th element of array age[]. */
```
/* Because, the first element of array is age[0], second is age[1], ith is age[i-1] and (i+1)th is age[i]. */

printf("%d",age[0]);

/* statement to print first element of an array. */

printf("%d",age[i]);

/* statement to print (i+1)th element of an array. */

Example of array in C programming

/* C program to find the sum marks of n students using arrays */

#include <stdio.h>

int main(){
    int marks[10],i,n,sum=0;
    printf("Enter number of students: ");
    scanf("%d",&n);
    for(i=0;i<n;++i){
        printf("Enter marks of student%d: ",i+1);
    }
    // Add code to calculate the sum here.
}

scanf("%d",&marks[i]);
    sum+=marks[i];
}

printf("Sum= %d",sum);
return 0;
}

Output

Enter number of students: 3
Enter marks of student1: 12
Enter marks of student2: 31
Enter marks of student3: 2
sum=45

Important thing to remember in C arrays

Suppose, you declared the array of 10 students. For example: arr[10]. You can use array members from arr[0] to arr[9]. But, what if you want to use elementarr[10], arr[13] etc. Compiler may not show error using these elements but, may cause fatal error during program execution.
C Programming Multidimensional Arrays

C programming language allows programmer to create arrays of arrays known as multidimensional arrays. For example:

```c
float a[2][6];
```

Here, `a` is an array of two dimension, which is an example of multidimensional array.

For better understanding of multidimensional arrays, array elements of above example can be thought of as below:

<table>
<thead>
<tr>
<th>row</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
<td>a[0][4]</td>
<td>a[0][5]</td>
</tr>
<tr>
<td>row 2</td>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
<td>a[1][4]</td>
<td>a[1][5]</td>
</tr>
</tbody>
</table>

Figure: Multidimensional Arrays

**Initialization of Multidimensional Arrays**

In C, multidimensional arrays can be initialized in different number of ways.

```c
int c[2][3]=\{\{1,3,0\}, \{-1,5,9\}\};
```

OR

```c
int c[][3]=\{\{1,3,0\}, \{-1,5,9\}\};
```
Initialization Of three-dimensional Array

double cprogram[3][2][4]= {
    {{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},
    {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},
    {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}
};

Suppose there is a multidimensional array arr[i][j][k][m]. Then this array can hold i*j*k*m numbers of data.

Similarly, the array of any dimension can be initialized in C programming.

**Example of Multidimensional Array In C**

Write a C program to find sum of two matrix of order 2*2 using multidimensional arrays where, elements of matrix are entered by user.

```c
#include <stdio.h>

int main() {
    float a[2][2], b[2][2], c[2][2];
```
int i,j;

printf("Enter the elements of 1st matrix\n");

/* Reading two dimensional Array with the help of two for loop. If there was an array of 'n' dimension, 'n' numbers of loops are needed for inserting data to array. */

for(i=0;i<2;++i)
    for(j=0;j<2;++j){
        printf("Enter a%d%d: ",i+1,j+1);
        scanf("%f",&a[i][j]);
    }

printf("Enter the elements of 2nd matrix\n");

for(i=0;i<2;++i)
    for(j=0;j<2;++j){
        printf("Enter b%d%d: ",i+1,j+1);
        scanf("%f",&b[i][j]);
    }

for(i=0;i<2;++i)
    for(j=0;j<2;++j){
        /* Writing the elements of multidimensional array using loop. */
        c[i][j]=a[i][j]+b[i][j]; /* Sum of corresponding elements of two arrays. */
    }

printf("\nSum Of Matrix:");
for(i=0;i<2;++i)
    for(j=0;j<2;++j){
        printf("%.1f\t",c[i][j]);
        if(j==1) /* To display matrix sum in order. */
            printf("\n");
    }

return 0;
}

Output

Enter the elements of 1st matrix

Enter a11: 2;

Enter a12: 0.5;

Enter a21: -1.1;

Enter a22: 2;

Enter the elements of 2nd matrix

Enter b11: 0.2;

Enter b12: 0;
Enter b21: 0.23;
Enter b22: 23;

Sum Of Matrix:

\[
\begin{pmatrix}
2.2 & 0.5 \\
-0.9 & 25.0
\end{pmatrix}
\]

C Programming Arrays and Functions

In C programming, a single array element or an entire array can be passed to a function. Also, both one-dimensional and multi-dimensional array can be passed to function as argument.

Passing One-dimensional Array In Function

C program to pass a single element of an array to function

```c
#include <stdio.h>

void display(int a)
{
  printf("%d",a);
}
```
int main()
{
    int c[]={2,3,4};
    display(c[2]);   //Passing array element c[2] only.
    return 0;
}

Output

4

Single element of an array can be passed in similar manner as passing variable to a function.

**Passing entire one-dimensional array to a function**

While passing arrays to the argument, the name of the array is passed as an argument(i.e, starting address of memory area is passed as argument).

Write a C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

```c
#include <stdio.h>

float average(float a[]);

int main()
{
    float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
    avg=average(c);   /* Only name of array is passed as argument. */
```
float average(float a[]){
    int i;
    float avg, sum=0.0;
    for(i=0;i<6;++i){
        sum+=a[i];
    }
    avg =(sum/6);
    return avg;
}

Output

Average age=27.08

Passing Multi-dimensional Arrays to Function

To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array

Example to pass two-dimensional arrays to function

#include
void Function(int c[2][2]);

int main(){
    int c[2][2], i, j;
    printf("Enter 4 numbers:\n");
    for(i=0;i<2;++i)
        for(j=0;j<2;++j){
            scanf("%d", &c[i][j]);
        }
    Function(c); /* passing multi-dimensional array to function */
    return 0;
}

void Function(int c[2][2]){
    /* Instead to above line, void Function(int c[][2]) is also valid */
    int i, j;
    printf("Displaying:\n");
    for(i=0;i<2;++i)
        for(j=0;j<2;++j)
            printf("%d\n", c[i][j]);
}

Output

Enter 4 numbers:
C Program to Calculate Average Using Arrays

This program takes n number of element from user (where, n is specified by user), stores data in an array and calculates the average of those numbers.

Source Code to Calculate Average Using Arrays
```c
#include <stdio.h>

int main()
{
    int n, i;
    float num[100], sum=0.0, average;
    printf("Enter the numbers of data: ");
    scanf("%d",&n);
    while (n>100 || n<=0)
    {
        printf("Error! number should in range of (1 to 100).\n");
        printf("Enter the number again: ");
        scanf("%d",&n);
    }
    for(i=0; i<n; ++i)
    {
        printf("%d. Enter number: ",i+1);
        scanf("%f",&num[i]);
        sum+=num[i];
    }
    average=sum/n;
    printf("Average = %.2f",average);
    return 0;
}
```

Output
Enter the numbers of data: 6

1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6

Average = 27.69

This program calculates the average if the number of data are from 1 to 100. If user enters value of \( n \) above 100 or below 100 then, while loop is executed which asks user to enter value of \( n \) until it is between 1 and 100.

C Program to Find the Largest Number Among Three Numbers

C Programming Arrays
In C programming, one of the frequently arising problem is to handle similar types of data. For example: If the user want to store marks of 100 students. This can be done by creating 100 variable individually but, this process is rather tedious and impracticable. These type of problem can be handled in C programming using arrays.

An array is a sequence of data item of homogeneous value(same type).

Arrays are of two types:

1. One-dimensional arrays
2. Multidimensional arrays

Declaration of one-dimensional array

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

For example:

```
int age[5];
```

Here, the name of array is age. The size of array is 5, i.e., there are 5 items(elements) of array age. All element in an array are of the same type (int, in this case).

Array elements

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:

```
int age[5];
```
Note that, the first element is numbered 0 and so on.

Here, the size of array `age` is 5 times the size of `int` because there are 5 elements.

Suppose, the starting address of `age[0]` is 2120d and the size of `int` be 4 bytes. Then, the next address (address of `a[1]`) will be 2124d, address of `a[2]` will be 2128d and so on.

**Initialization of one-dimensional array:**

Arrays can be initialized at declaration time in this source code as:

```c
int age[5]={2,4,34,3,4};
```

It is not necessary to define the size of arrays during initialization.

```c
int age[]= {2,4,34,3,4};
```

In this case, the compiler determines the size of array by calculating the number of elements of an array.

**Accessing array elements**

In C programming, arrays can be accessed and treated like variables in C.

For example:
scanf("%d", &age[2]);
/* statement to insert value in the third element of array age[]. */

scanf("%d", &age[i]);
/* Statement to insert value in (i+1)th element of array age[]. */
/* Because, the first element of array is age[0], second is age[1], ith is age[i-1] and (i+1)th is age[i]. */

printf("%d", age[0]);
/* statement to print first element of an array. */

printf("%d", age[i]);
/* statement to print (i+1)th element of an array. */

Example of array in C programming

/* C program to find the sum marks of n students using arrays */

#include <stdio.h>

int main(){

```c
int marks[10], i, n, sum = 0;
printf("Enter number of students: ");
scanf("%d", &n);
for (i = 0; i < n; ++i){
    printf("Enter marks of student%d: ", i + 1);
    scanf("%d", &marks[i]);
    sum += marks[i];
}
printf("Sum = %d", sum);
return 0;
}
```

**Output**

Enter number of students: 3

Enter marks of student1: 12

Enter marks of student2: 31

Enter marks of student3: 2

sum=45

**Important thing to remember in C arrays**

Suppose, you declared the array of 10 students. For example: `arr[10]`. You can use array members from `arr[0]` to `arr[9]`. But, what if you want to use elements `arr[10]`, `arr[13]` etc. Compiler may not show error using these elements but, may cause fatal error during program execution.
C Programming Multidimensional Arrays

C programming language allows programmer to create arrays of arrays known as multidimensional arrays. For example:

```c
float a[2][6];
```

Here, `a` is an array of two dimension, which is an example of multidimensional array.

For better understanding of multidimensional arrays, array elements of above example can be thought of as below:

<table>
<thead>
<tr>
<th>row 1</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
<td>a[0][4]</td>
<td>a[0][5]</td>
<td></td>
</tr>
<tr>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
<td>a[1][4]</td>
<td>a[1][5]</td>
<td></td>
</tr>
</tbody>
</table>

**Figure: Multidimensional Arrays**

Initialization of Multidimensional Arrays

In C, multidimensional arrays can be initialized in different number of ways.

```c
int c[2][3]={ {1,3,0}, {-1,5,9} };  

OR

int c[][3]={ {1,3,0}, {-1,5,9} };  
```
OR

```c
int c[2][3]={1,3,0,-1,5,9};
```

**Initialization Of three-dimensional Array**

```c
double cprogram[3][2][4]= {
    {{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},
    {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},
    {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}
};
```

Suppose there is a multidimensional array `arr[i][j][k][m]`. Then this array can hold `i*j*k*m` numbers of data.

Similarly, the array of any dimension can be initialized in C programming.
**Linear search in c programming:** The following code implements linear search (Searching algorithm) which is used to find whether a given number is present in an array and if it is present then at what location it occurs. It is also known as sequential search. It is very simple and works as follows: We keep on comparing each element with the element to search until the desired element is found or list ends. Linear search in c language for multiple occurrences and using function.

```
#include <stdio.h>
int main(){
    int array[100], search, c, n;

    printf("Enter the number of elements in array\n");
    scanf("%d", &n);

    printf("Enter %d integer(s)\n", n);

    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);

    printf("Enter the number to search\n");
    scanf("%d", &search);

    for (c = 0; c < n; c++)
    {
        if (array[c] == search)  /* if required element found */
        {
            printf("%d is present at location %d.\n", search, c+1);
            break;
        }
    }

    if (c == n)
        printf("%d is not present in array.\n", search);

    return 0;
}
```
C program to find maximum element in array

This code finds the maximum or largest element present in an array. It also prints the location or index at which the maximum element occurs in the array. This can also be done by using pointers (see both codes). The algorithm to find maximum is first we assume that the maximum element occurs at the beginning of the array and stores that value in a variable. Then we compare it with other array elements one by one, if any element is greater than our assumed maximum then maximum value and index at which it occurs is updated. Similarly, we can find the minimum element in an array.

```c
#include <stdio.h>

int main()
{
    int array[100], maximum, size, c, location = 1;

    printf("Enter the number of elements in array\n");
    scanf("%d", &size);

    printf("Enter %d integers\n", size);

    for (c = 0; c < size; c++)
        scanf("%d", &array[c]);

    maximum = array[0];

    for (c = 1; c < size; c++)
    {
        if (array[c] > maximum)
        {
            maximum = array[c];
            location = c+1;
        }
    }

    return 0;
}
```
C program to find minimum element in array

C code to find minimum or smallest element present in an array. It also prints the location or index at which minimum element occurs in array. This can also be done by using pointers (see both the codes). Our algorithm first assumes first element as minimum and then compare it with other elements if an element is smaller than it then it becomes the new minimum and this process is repeated till complete array is scanned.

C programming code

```c
#include <stdio.h>

int main() {
    int array[100], minimum, size, c, location = 1;

    printf("Enter the number of elements in array\n");
    scanf("%d", &size);

    printf("Enter %d integers\n", size);
```
```c
for ( c = 0 ; c < size ; c++ )
    scanf("%d", &array[c]);

minimum = array[0];

for ( c = 1 ; c < size ; c++ )
{
    if ( array[c] < minimum )
    {
        minimum = array[c];
        location = c+1;
    }
}

printf("Minimum element is present at location %d and it's value is %d.\n", location, minimum);
return 0;)
```

If minimum occurs two or more times times in array then index at which it occurs first is printed or minimum value at smallest index. You can modify this code this code to print largest index at which minimum occur. You can also store all indices at which minimum occur in an array.

Output of program:

![Image showing the output of the program](image.png)

**Matrix multiplication in c**

Matrix multiplication in c language: c program to multiply matrices (two dimensional array), this program multiplies two matrices which will be entered by the user. Firstly user will enter the order of a matrix. If the entered orders of two matrix is such that they can't be multiplied
then an error message is displayed on the screen. You have already studied the logic to multiply them in Mathematics.

Matrix multiplication in c language

```c
#include <stdio.h>

int main()
{
    int m, n, p, q, c, d, k, sum = 0;
    int first[10][10], second[10][10], multiply[10][10];

    printf("Enter the number of rows and columns of first matrix\n");
    scanf("%d%d", &m, &n);
    printf("Enter the elements of first matrix\n");

    for (c = 0; c < m; c++)
        for (d = 0; d < n; d++)
            scanf("%d", &first[c][d]);

    printf("Enter the number of rows and columns of second matrix\n");
    scanf("%d%d", &p, &q);

    if (n != p)
        printf("Matrices with entered orders can’t be multiplied with each other.\n");
    else
    {
        printf("Enter the elements of second matrix\n");

        for (c = 0; c < p; c++)
            for (d = 0; d < q; d++)
                scanf("%d", &second[c][d]);

        for (c = 0; c < m; c++) {
            for (d = 0; d < q; d++) {
                for (k = 0; k < p; k++) {
                    sum = sum + first[c][k]*second[k][d];
                }

                multiply[c][d] = sum;
                sum = 0;
            }
        }
    }
```

In C programming, a single array element or an entire array can be passed to a function. Also, both one-dimensional and multi-dimensional array can be passed to function as argument.

**Passing One-dimensional Array In Function**

**C program to pass a single element of an array to function**

```c
#include <stdio.h>

void display(int a) {
    printf("%d", a);
}
```


```c
int main()
{
    int c[]={2,3,4};
    display(c[2]);      //Passing array element c[2] only.
    return 0;
}
```

Output

```
4
```

Single element of an array can be passed in similar manner as passing variable to a function.

**Passing entire one-dimensional array to a function**

While passing arrays to the argument, the name of the array is passed as an argument,i.e, starting address of memory area is passed as argument).

Write a C program to pass an array containing age of person to a function. This function should find average age and display the average age in main function.

```c
#include <stdio.h>

float average(float a[]);

int main()
{
    float avg, c[]={23.4, 55, 22.6, 3, 40.5, 18};
    avg=average(c);      /* Only name of array is passed as argument. */
```
```c
float average(float a[]){
    int i;
    float avg, sum=0.0;
    for(i=0; i<6; ++i){
        sum+=a[i];
    }
    avg =(sum/6);
    return avg;
}
```

Output

Average age=27.08

Passing Multi-dimensional Arrays to Function

To pass two-dimensional array to a function as an argument, starting address of memory area reserved is passed as in one dimensional array

Example to pass two-dimensional arrays to function

```c
#include
```
void Function(int c[2][2]);

int main(){
    int c[2][2],i,j;
    printf("Enter 4 numbers:\n");
    for(i=0;i<2;++i)
        for(j=0;j<2;++j){
            scanf("%d",&c[i][j]);
        }
    Function(c);    /* passing multi-dimensional array to function */
    return 0;
}

void Function(int c[2][2]){    /* Instead to above line, void Function(int c[][2]){ is also valid */
    int i,j;
    printf("Displaying:\n");
    for(i=0;i<2;++i)
        for(j=0;j<2;++j)
            printf("%d\n",c[i][j]);
}

Output

Enter 4 numbers:
In C programming, array of character are called strings. A string is terminated by null character /0. For example:

"c string tutorial"

Here, "c string tutorial" is a string. When, compiler encounters strings, it appends null character at the end of string.
Declaration of strings

Strings are declared in C in similar manner as arrays. Only difference is that, strings are of `char` type.

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

Strings can also be declared using pointer.

```c
char *p
```

Initialization of strings

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

```c
char c[]="abcd";

OR,

char c[5]="abcd";

OR,

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

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

String can also be initialized using pointers

char *c="abcd";

Reading Strings from user.

Reading words from user.

char c[20];

scanf("%s",c);

String variable c can only take a word. It is because when white space is encountered, the scanf() function terminates.

Write a C program to illustrate how to read string from terminal.

#include <stdio.h>

int main(){
    char name[20];

    printf("Enter name: ");
```c
#include <stdio.h>

int main(){

    char name[30], ch;

    int i=0;

    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 will ignore Ritchie because, `scanf()` function takes only string before the white space.

**Reading a line of text**

**C program to read line of text manually.**

```c
#include <stdio.h>

int main(){

    char name[30], ch;

    int i=0;

    printf("Enter name: ");
```
```c
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;
}
```

This process to take string is tedious. There are predefined functions `gets()` and `puts` in C language to read and display string respectively.

```c
int main(){
    char name[30];
    printf("Enter name: ");
    gets(name); //Function to read string from user.
}
Both, the above program has same output below:

Output

Enter name: Tom Hanks

Name: Tom Hanks

Passing Strings to Functions

String can be passed to function in similar manner as arrays as, string is also an array. Learn more about passing array to a function.
Here, string c is passed from main() function to user-defined function Display(). In function declaration, ch[] is the formal argument.

**String handling functions**

You can perform different type of string operations manually like: finding length of string, concatenating(joining) two strings etc. But, for programmers ease, many library function are defined under header file <string.h> to handle these commonly used talk in C programming. You will learn more about string hadling function in next chapter.
String Manipulations In C Programming Using Library Functions

Strings are often needed to be manipulated by programmer according to the need of a problem. All string manipulation can be done manually by the programmer but, this makes programming complex and large. To solve this, the C supports a large number of string handling functions.

There are numerous functions defined in "string.h" header file. Few commonly used string handling functions are discussed below:

<table>
<thead>
<tr>
<th>Function</th>
<th>Work of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>strlen()</td>
<td>Calculates the length of string</td>
</tr>
<tr>
<td>strcpy()</td>
<td>Copies a string to another string</td>
</tr>
<tr>
<td>strcat()</td>
<td>Concatenates(joins) two strings</td>
</tr>
<tr>
<td>strcmp()</td>
<td>Compares two string</td>
</tr>
<tr>
<td>strlwr()</td>
<td>Converts string to lowercase</td>
</tr>
<tr>
<td>strupr()</td>
<td>Converts string to uppercase</td>
</tr>
</tbody>
</table>

Strings handling functions are defined under "string.h" header file, i.e, you have to include the code below to run string handling functions.

```c
#include <string.h>
```
gets() and puts()

Functions gets() and puts() are two string functions to take string input from user and display string respectively as mentioned in previous chapter.

```c
#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;
}
```

Though, gets() and puts() function handle string, both these functions are defined in "stdio.h" header file.
Matrix mult

Example: Find the product $AB$ where $A$ and $B$ are matrices given by

1.

$$A = \begin{bmatrix} 2 & -3 \\ 4 & 5 \\ 6 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$$

Solution

The product $AB$ is defined since $A$ is a $3 \times 2$ matrix and has 2 columns and $B$ is a $2 \times 2$ matrix and has 2 rows. To find elements of the product $C = AB$, multiply each row of $A$ by each column of $B$.

$$C = AB = \begin{bmatrix} [2 & -3] [1 & 0] \\ [4 & 5] [-2 & 1] \\ [6 & 0] \end{bmatrix}$$

$$= \begin{bmatrix} (2)(1) + (-3)(-2) & (2)(0) + (-3)(1) \\ (4)(1) + (5)(-2) & (4)(0) + (5)(1) \\ (6)(1) + (0)(-2) & (6)(0) + (0)(1) \end{bmatrix} = \begin{bmatrix} 8 & -3 \\ -6 & 5 \\ 6 & 0 \end{bmatrix}$$

End of the course