Course Code: - CSM-6151
Course Name: - Programming with 'C' & Lab

MASTER OF COMPUTER APPLICATIONS (MCA)

CONTENT

Programs:

1. Write a C program to find the roots of a quadratic equation.
2. Write a C program to find the total no. of digits and the sum of individual digits of a positive integer.
3. Write a C program to generate the Fibonacci sequence of the first N numbers.
4. Write a C program to compute the area of a circle, a Square, and a Rectangle when all the dimensions are given.
5. Write a C program to input two matrices and perform matrix multiplication on them.
6. Write a C program to check whether the given string is a palindrome or not without using Library functions.
7. Write a C program to count the number of lines and words in a given file.
8. Write a C program to generate prime numbers in a given range using a user-defined function.
9. Write a C program to find the factorial of a given number using a recursive function.
10. Write a C program to maintain a record of n student details using an array of structures with four fields - Roll number, Name, Marks, and Grade. Calculate the Grade according to the following conditions.

Marks	Grade
>= 80	A
>= 60	B
>= 50	C
>= 40	D
< 40	E

Print the details of the student, given the student's roll number as input.

CSM -6151 PROGRAMMING WITH ‘C’ LAB

COURSE INTRODUCTION

Welcome to the world of C programming! This book serves as your gateway into the foundational aspects of computer programming using the C language. C, developed by Dennis Ritchie in the early 1970s, remains one of the most influential and widely used programming languages, serving as the bedrock for numerous modern languages and systems.

This book is on C programming. We focus on problem solving using the language and present standard programming techniques such as alternation, iteration, and recursion. We will look at the fundamentals of software engineering principles such as modularization, commenting, and naming conventions that aid in team collaboration and development.

OVERVIEW

In this book, we'll embark on an exciting journey to understand the core concepts and principles that underpin C programming. From mastering the syntax to comprehending powerful concepts like pointers and memory management, each module will equip you with the tools necessary to write efficient, robust, and scalable programs.

What You'll Learn:

- **Fundamentals:** You'll begin with the basics, learning about variables, data types, operators, and control structures that form the building blocks of C programming.
- **Functions and Modularity:** Dive into the realm of functions, understanding their role in code organization, passing arguments, return values, and utilizing libraries.
- **Memory Management:** Explore the intricate world of

memory, understanding pointers, dynamic memory allocation, and their significance in optimizing code efficiency.

- **Advanced Concepts:** We'll cover more advanced topics like file handling, structures, and preprocessor directives, elevating your programming prowess.

How You'll Learn:

Through a blend of comprehensive lectures, hands-on coding exercises, quizzes, and real-world examples, you'll grasp not just the theory but also the practical applications of C programming. The book is designed to encourage active participation, allowing you to code alongside the lessons and solidify your understanding through practice.

Who Is This Book For:

Whether you're a novice programmer eager to start your journey or an experienced developer seeking to deepen your understanding of low-level programming, this book caters to individuals at all levels. It's ideal for students, hobbyists, professionals, and anyone passionate about honing their programming skills.

Conclusion:

By the end of this book, you'll emerge with a solid foundation in C programming, equipped to write efficient algorithms, create versatile applications, and approach more complex programming challenges with confidence. Get ready to unlock the potential of C programming and embark on a transformative learning experience that will shape your programming journey for years to come!

This introduction aims to provide a glimpse into the learning approach, target audience, and the potential impact it can have on a learner's programming abilities.

CSM – 6151: PROGRAMMING WITH ‘C’ LAB

Programming Lab

OVERVIEW OF THE LAB

This lab course offers hands-on experience aimed at practical application. Participants have finished the BCA-151 Programming Principles and Algorithms Labs support course, which covers C programming examples across Windows, UNIX, and DOS systems. Each session concludes with a series of programming problems for practice. It's essential to thoroughly review the program documentation regulations and adhere to the general guidelines provided.

Program development steps:

This program development involves a series of sequential steps essential for creating in a high-level language and converting it *into machine-level language:*

1. Drafting and refining the software design.
2. Linking the application with required library modules.
3. Compiling the software.
4. Executing the program for operation.

This lab course will discuss the separate step compilation process of the language C.

OBJECTIVES

After completing this lab course, you will be able to:

- The goal is to guide learners in comprehending the problem-solving rationale and algorithm formulation process.
- This includes crafting a corresponding flowchart and comprehending the syntax and framework of C programming.
- Proficiency in procedural language programming is emphasized, covering the methods for assembling, connecting, and resolving issues in C code.
- The primary aim is to impart a foundational comprehension of C language programming to students. This encompasses teaching them problem-solving strategies and proficient writing techniques in C programming.
- The curriculum covers fundamental concepts such as functions, pointers, file handling, structures, loops, and arrays, ensuring a comprehensive understanding of these key components in programming.

OVERALL DIRECTIONS

To try each of the tasks and challenges listed in the list, session by session.

You can ask the responsible lab teacher for help completing the lab exercises. The lab teacher is clearly not expected to provide you with solutions to the assignments as they are credit-based, but you are welcome to ask questions about the C language or any technical issues.

You should put comments (text in between /*... */ delimiters) above every function in the code, including the main function, for every application. A description of the function that has been developed, its goal, the significance of the parameter it uses, and the meaning of the return result, if any, should also be included.

Prior to the primary function's source code, comprehensive explanations outlining the purpose of the program will be incorporated within the comment block. These explanations will elucidate the program's objectives and functionalities. Throughout the code, relevant comments will be strategically placed to enhance readability and understanding.

The C program will strictly adhere to the ANSI standard for the language, ensuring compatibility and conformity. It will be developed as a generic and interactive application, meticulously documented with real input and output data to facilitate comprehension.

To maintain integrity, submissions that appear derivative, stemming from multiple sources but exhibiting remarkable similarities, will not be considered. It is strongly advised against replicating or mimicking someone else's work to ensure individuality and authenticity in submissions.

Your responsibility includes creating a separate directory inaccessible to others for storing all programs to ensure confidentiality. Keeping an Observation Book and Lab Record is mandatory. The lab manual provides a session-wise list of programs, and it's essential to prepare algorithms and record programs in the Observation Book prior to each session.

During lab hours, dedicate time to executing, testing, and enhancing programs for desired outputs. Upon completing a lab exercise, approach a lab instructor or in-charge for evaluation and signature in the Observation Book.

Lab assignments should be submitted in the form of a comprehensive Lab Record. This record should encompass algorithms, program codes with comments, and outputs for various inputs provided, ensuring a thorough documentation of the work completed.

STRUCTURE OF ‘C’ PROGRAM

A 'C' program is constructed from multiple instructions, each written as an individual statement. It commences with the main function enclosed in opening braces, signifying its initiation. This is succeeded by variable and constant declarations, succeeded by statements encompassing input and output operations.

The structure of a 'C' program typically involves several sections, as outlined below:

1. Initialization: Begins with the 'main' function, the entry point of execution.
2. Variable and Constant Declarations: Defines variables and constants necessary for the program.
3. Statements: Includes instructions for data processing, involving input/output operations and logic implementation.

These sections collectively form the structure of a 'C' program, organizing the flow of operations:

DOCUMENTATION SECTION
LINK SECTION
DEFINITION SECTION
GLOBAL DECLARATION SECTION
Main() Function Section
{
    Declaration part
    Executable part
}
SUBPROGRAM SECTION
    User defined function

SALIENT FEATURES OF C

C language boasts several defining characteristics that have propelled its popularity within the programming landscape, many of which were thoroughly covered during the BCA-151 Problem Solving and Programming course:

1. Small Size: C's concise nature allows for efficient coding without unnecessary overheads.

2. Extensive Use of Function Calls: Its modular approach leverages functions for code organization and reusability.

3. Structured Language: Encourages organized and systematic programming practices, enhancing readability and maintenance.

4. Low-Level (Bitwise) Programming Availability: Offers direct access to memory and bitwise operations for optimized code implementation.

5. Pointer Implementation: Extensively employs pointers for memory management, arrays, structures, and functions, enabling intricate data manipulation.

6. High-Level Constructs: Provides high-level constructs for abstraction, simplifying complex operations.

7. Handling Low-Level Activities: Allows direct manipulation of hardware, making it suitable for system programming and embedded systems.

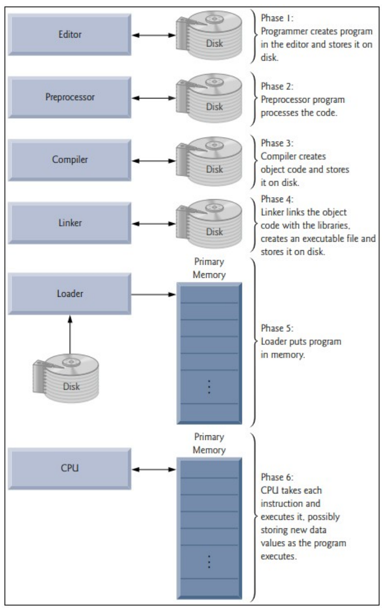
8. Efficient Program Output: Produces highly efficient programs due to its close-to-hardware approach.

9. Cross-Platform Compilation: Offers portability, allowing compilation on various computer architectures, contributing to its versatility and widespread use."

These features collectively contribute to C's robustness, flexibility, and efficiency, making it a prominent choice for diverse programming needs.

‘C’ PROGRAM DEVELOPMENT ENVIRONMENT

C systems generally consist of several parts: a program-development environment, the language and the C Standard Library. Explain the following typical C development environment:


Editor
Disk
Phase 1:
Programmer creates program in the editor and stores it on disk.
Preprocessor
Disk
Phase 2:
Preprocessor program processes the code.
Compiler
Disk
Phase 3:
Compiler creates object code and stores it on disk.
Linker
Disk
Phase 4:
Linker links the object code with the libraries, creates an executable file and stores it on disk.
Loader
Disk
Primary Memory
Phase 5:
Loader puts program in memory.
CPU
Primary Memory
Phase 6:
CPU takes each instruction and executes it, possibly storing new data values as the program executes.

C programs typically go through six phases to be executed. These are: edit, preprocess, compile, link, load and execute.

Phase – I: Creating a Program

Visual Studio Code stands out as one of the most widely utilized code editors and integrated development environments (IDEs) developed by Microsoft. It serves as a versatile platform for coding in various programming languages, fostering the creation and optimization of codebases while facilitating efficient debugging. Notably, Visual Studio Code boasts cross-platform compatibility, running seamlessly on Windows, macOS, and Linux operating systems.

The editor's popularity extends globally, including in India, where it has gained widespread adoption. Its user-friendly interface and extensive language support, encompassing languages such as C, C++, Java, Python, JavaScript, React, and Node.js, contribute to its broad appeal. Visual Studio Code distinguishes itself by offering a rich ecosystem of in-app extensions tailored for diverse programming languages, enabling users to tailor their coding environment to specific needs.

One of the noteworthy features of Visual Studio Code is its visually appealing and dynamic user interface, complemented by a sophisticated night mode that enhances the coding experience. The editor facilitates a smooth coding process by providing users with auto-complete code suggestions, streamlining the writing of code and enhancing overall productivity.

In conclusion, Visual Studio Code has secured its position as a premier code editor and IDE due to its versatility, extensive language support, user-friendly interface, and a plethora of features, making it a top choice for programmers across the globe, including in India.

Phase–II&III: Preprocessing and Compiling a ‘C’ Program

The compiler translates the C program into machine-language code, also known as object code. Before the translation phase, a preprocessor program in the C system automatically executes. This preprocessor adheres to special commands known as preprocessor directives, instructing specific manipulations to be carried out on the program before compilation. These manipulations commonly involve including other files within the file being compiled and performing text replacements.

The compiler converts the C program into machine-language code. Syntax errors occur when the compiler encounters a statement that violates the language's rules and cannot be recognized. In response, the compiler issues an error message, aiding in locating and rectifying the erroneous statement. It's worth noting that the wording for error messages issued by the compiler isn't standardized by the C Standard, leading to potential variations in error messages across different systems. These errors are commonly referred to as compile errors or compile-time errors.

Install Visual Studio Code on Windows

To Install Visual Studio Code on a Windows System, follow these steps:

1. Download Visual Studio Code:

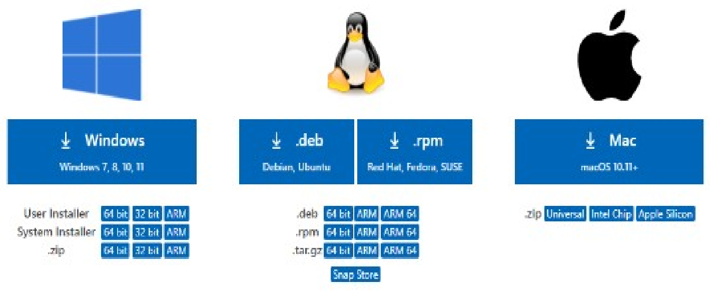
- Open your web browser and navigate to the official Visual

Studio Code website: https://code.visualstudio.com/.

- Click on the "Download for Windows" button.

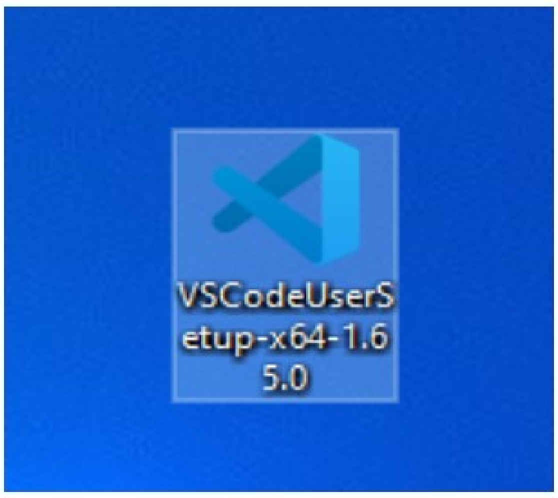
2. Run the Installer:

- Once the installer executable (.exe) is downloaded, locate the file (usually in the Downloads folder) and double-click on it to run the installer.



3. Begin Installation:

- The installer will prompt you to confirm that you want to install Visual Studio Code. Click "Yes" or "Run" to proceed.

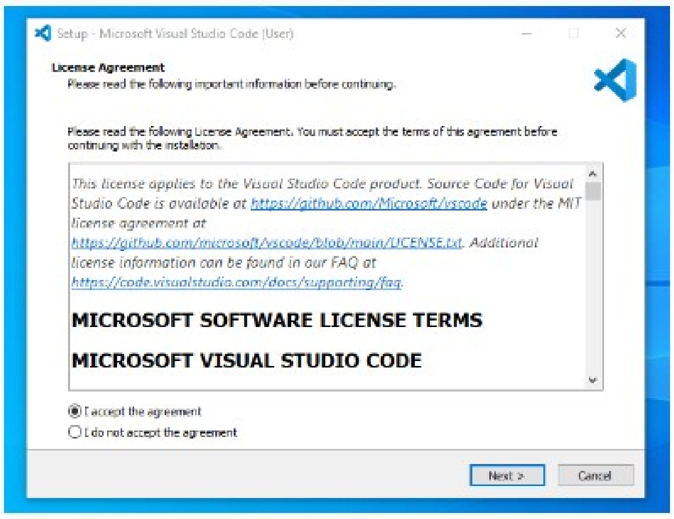


4. Choose Setup Options:

- The installer will provide various setup options. You can choose the default settings or customize them according to your preferences. For most users, the default options are sufficient.

5. Select Additional Tasks:

- The installer may offer additional tasks, such as creating desktop shortcuts or adding entries to the PATH environment variable. Choose the options that suit your preferences.



Setup - Microsoft Visual Studio Code (User)

License Agreement

Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

This license applies to the Visual Studio Code product. Source Code for Visual Studio Code is available at https://github.com/Microsoft/vscode under the MIT license agreement at https://github.com/microsoft/vscode/blob/main/LICENSE.txt. Additional license information can be found in our FAQ at https://code.visualstudio.com/docs/supporting/faq.

MICROSOFT SOFTWARE LICENSE TERMS

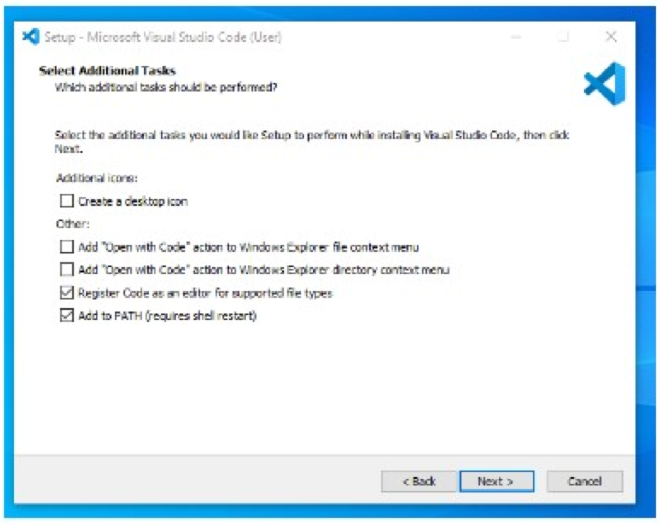
MICROSOFT VISUAL STUDIO CODE

[✓ I accept the agreement]

I do not accept the agreement

6. Install:

- Click the "Install" button to initiate the installation process. The installer will copy the necessary files and set up Visual Studio Code on your system.



Setup - Microsoft Visual Studio Code (User)

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Visual Studio Code, then click Next.

Additional icons:

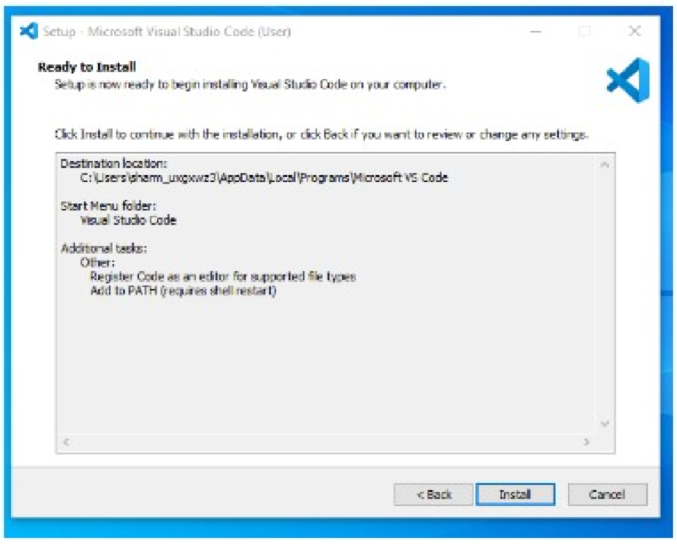
- [ ] Create a desktop icon

Other:

- [ ] Add "Open with Code" action to Windows Explorer file context menu
- [ ] Add "Open with Code" action to Windows Explorer directory context menu
- [x] Register Code as an editor for supported file types
- [x] Add to PATH (requires shell restart)

7. Complete Installation:

- Once the installation is complete, you will see a confirmation message. You can choose to launch Visual Studio Code immediately by leaving the corresponding option checked.



Setup - Microsoft Visual Studio Code (User)

Ready to Install

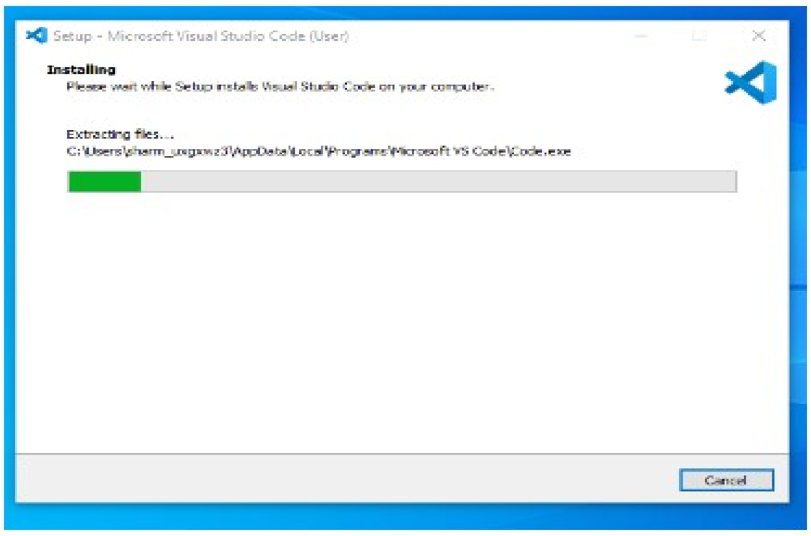
Setup is now ready to begin installing Visual Studio Code on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
    C:\Users\sharm_uxgxwz3\AppData\Local\Programs\Microsoft VS Code

Start Menu folder:
    Visual Studio Code

Additional tasks:
    Other:
      Register Code as an editor for supported file types
      Add to PATH (requires shell restart)
```

8. Launch Visual Studio Code:

- If you didn't choose to launch it during the installation, you can find the Visual Studio Code shortcut on your desktop or in the Start menu. Double-click on the shortcut to open Visual Studio Code.



Setup - Microsoft Visual Studio Code (User)

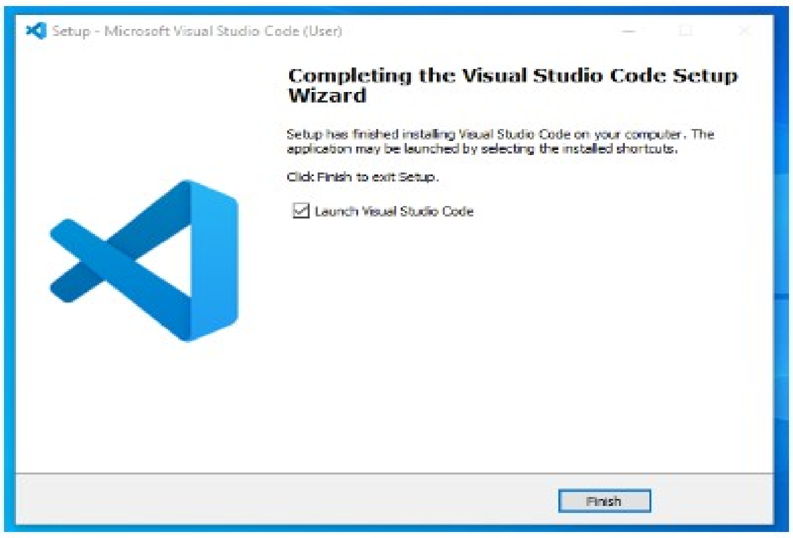
Installing

Please wait while Setup installs Visual Studio Code on your computer.

```
Extracting files...
C:\Users\sharm_uxgxwz3\AppData\Local\Programs\Microsoft VS Code\Code.exe
```

9. Update Extensions (Optional):

- After launching Visual Studio Code, you may want to explore and install extensions based on your programming needs. You can access the Extensions view by clicking on the Extensions icon in the Activity Bar on the side.


Setup - Microsoft Visual Studio Code (User)

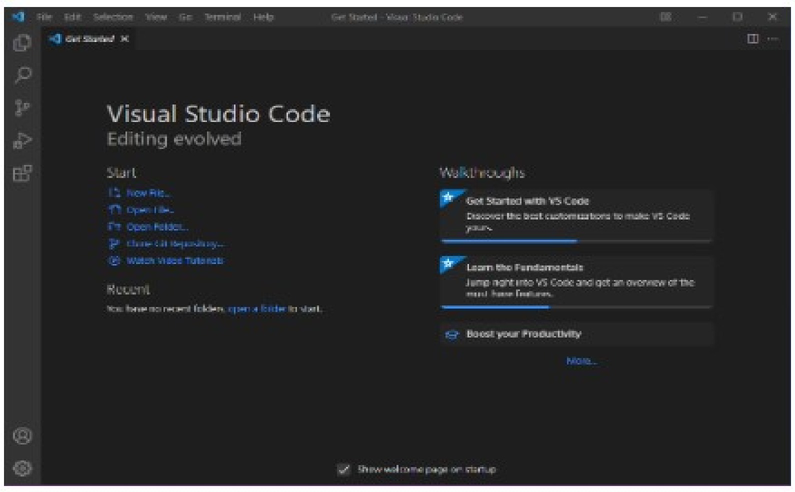
Completing the Visual Studio Code Setup Wizard

Setup has finished installing Visual Studio Code on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

☑ Launch Visual Studio Code

That's it! You have successfully installed Visual Studio Code on your Windows system, and ready to start coding in your preferred programming languages.



You have no recent folders, open a folder to start.

Discover the best customizations to make VS Code yours.

Jump right into VS Code and get an overview of the must-have features.

successfully installed Visual Studio Code **on our Windows system.**

HOW TO DESIGN/DEVELOP PROGRAM

Steps involved in program development:

To develop the program in high level language and translate it into machine level language following steps have to be practised.

1. Writing and editing the program.
2. Linking the program with the required library modules.
3. Compiling the program.
4. Executing the program.

Algorithm:

It is a method of representing the step by step process for solving a problem. Each step is called an instruction.

Characteristics of algorithm are:

Finiteness: It terminates with finite number of steps.

Definiteness: Each step of algorithm is exactly defined.

Effectiveness: All the operations used in the algorithm can be performed exactly in a fixed duration of time.

Input: An algorithm must have an input before the execution of program begins.

Output: An algorithm has one or more outputs after the execution of the program.

Example of algorithm to find sum of two numbers:

```
Step1: BEGIN
Step2: READ a, b
Step3: ADD a and b and store in variable c
Step4: DISPLAY c
Step5: STOP
```

STRUCTURE OF 'C' PROGRAM

C program is a collection of several instructions where each instruction is written as a separate statement. The C program starts with a main function followed by the opening braces which indicates the start of the function. Then follows the variable and constant declarations which are followed by the statements that include input and output statements.

C program may contain one or more sections as:

```
DOCUMENTATION SECTION
LINK SECTION
DEFINITION SECTION
GLOBAL DECLARATION SECTION
Main() Function section
{
        Declaration part
        Executable part
}
SUBPROGRAM SECTION
        User defined functions
```

Example:

Write a C program to find the sum and average of three numbers.

Algorithm:

Step 1: Start

Step 2: Declare variables num1, num2, num3 and sum, average.

Step 3: Read values num1, num2, num3.

Step 4: Add num1, num2, num3 and assign the result to sum.

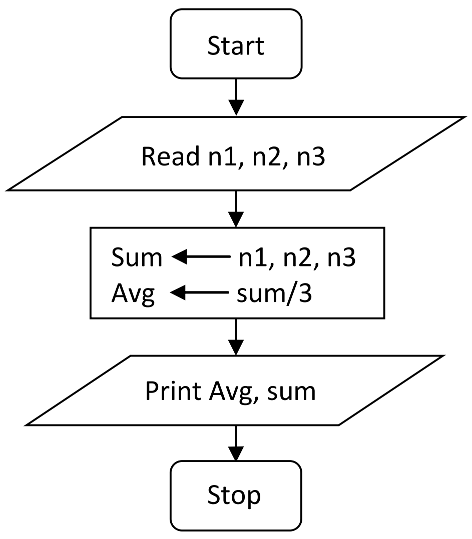
sum $\longleftarrow$ num1 + num2 + num3

average $\longleftarrow$ sum/3

Step 5: Display sum and average

Step 6: Stop

Flow Chart:



Program:

```
#include<stdio.h>
void main( )
{
int a,b,c;
int sum,average;
printf("Enter any three integers: ");
scanf("%d%d %d",&a,&b,&c);
sum = a+b+c;
average=sum/3;
printf("Sum and average of three integers: %d %d",sum,average);
return 0;
}
```

INPUT: Enter any three integers:2 4 5

OUTPUT: Sum and average of three integers: 11 3

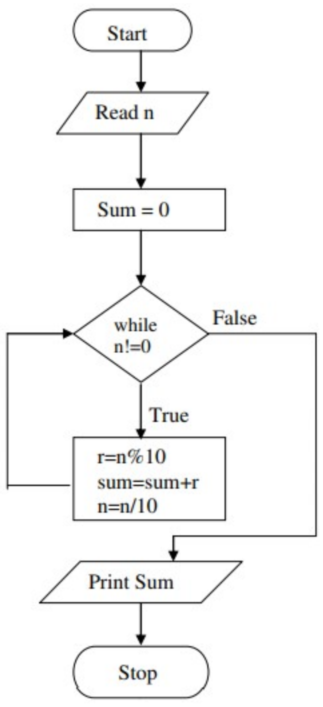
Example:

Write a C program to find the sum of individual digits of positive integer.

Algorithm:

```
Step 1: Start
Step 2: Read
Step 3: Initialize sum ← 0
Step 4: while(n!=0)
        Begin
Step 5: r ← n%10
Step 6: Sum ← Sum+r
Step 7: n ← n/10
        End
Step 8: Print “sum”
Step 9: Stop
```

Flow Chart:



Program:

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int n,r,sum=0;
clrscr();
printf("ENTER A POSITIVE INTEGER \n");
scanf("%d",&n);
while(n!=0)
{
r=n%10;
sum=sum+r;
n=n/10;
}
printf("THE SUMOF INDIVIDUAL DIGITS OF A POSITIVE INTEGER IS..%d",sum); getch();
}
```

INPUT: ENTER A POSITIVE INTEGER 5 3 2 1

OUTPUT: THE SUM OF INDIVIDUAL DIGITS OF A POSITIVE INTEGER IS..11

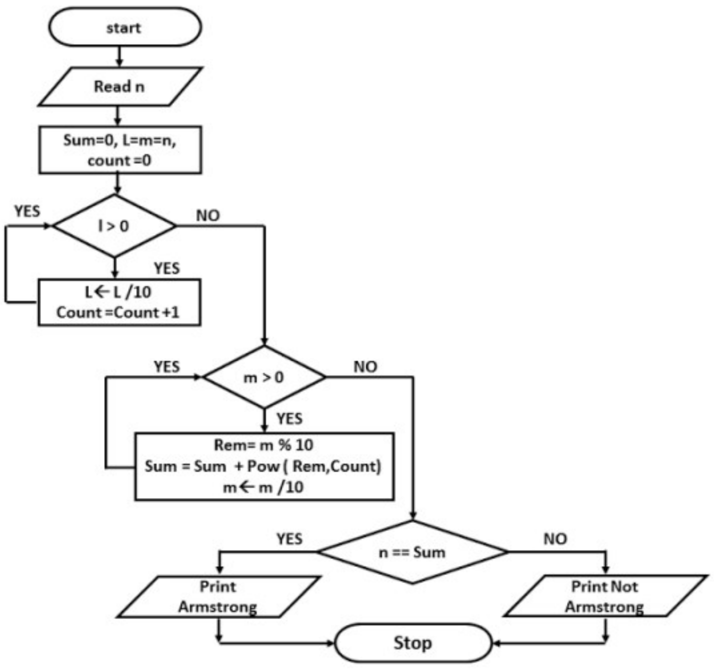
Example:

Write a C program to check whether given number is Armstrong Number or Not.

Algorithm:

```
Step 1: Start
Step 2: Read n
Step 3: assign sum
Step 4: if m>0 repeat
        Step 4.1: m ⟵ m/10
        Step 4.2: count++
        Step 4.3: until the condition fail
Step5: if I>0 repeat step 4 until condition fail
        Step 5.1:rem ⟵ I%10
        Step 5.2:sum ⟵ sum+pow(rem,count)
        Step 5.3:I⟵I/10
Step 6:if n=sum print Armstrong otherwise print not armstrong
Step 7:stop
```

Flow Chart:



Program:

```
#include<stdio.h>
void main()
{
int n, n1, rem, num=0;
printf("Enter a positive integer: ");
scanf("%d", &n); n1=n;
while(n1!=0)
{
rem=n1%10;
num+=rem*rem*rem;
n1/=10;
}
if(num==n)
printf("%d is an Armstrong number.",n);
else printf("%d is not an Armstrong number.",n);
}
```

Input: Enter a positive integer: 371

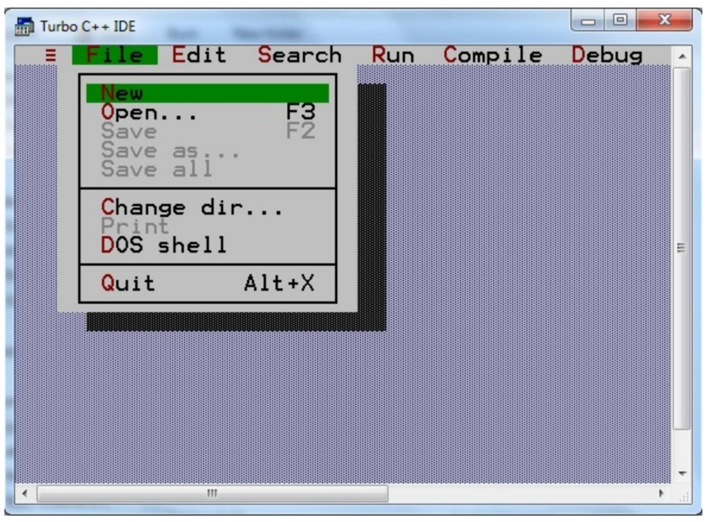
Output: 371 is an Armstrong number.

COMPILE AND RUN 'C' PROGRAM

To compile and run a C language program, you need a C compiler. A **compiler** is software that is used to compile and execute programs. To set up a C language compiler on your computer/laptop, there are two ways:

1. Download a full-fledged IDE like Turbo C++ or Microsoft Visual C++ or DevC++, which comes along with a C language compiler.
2. Or, you can use any text editor to edit the program files and download the C compiler separately and then run the C program using the command line.

If you haven't already installed an IDE for the C language - Follow this step-by-step guide to **Install Turbo C++ for C Language**

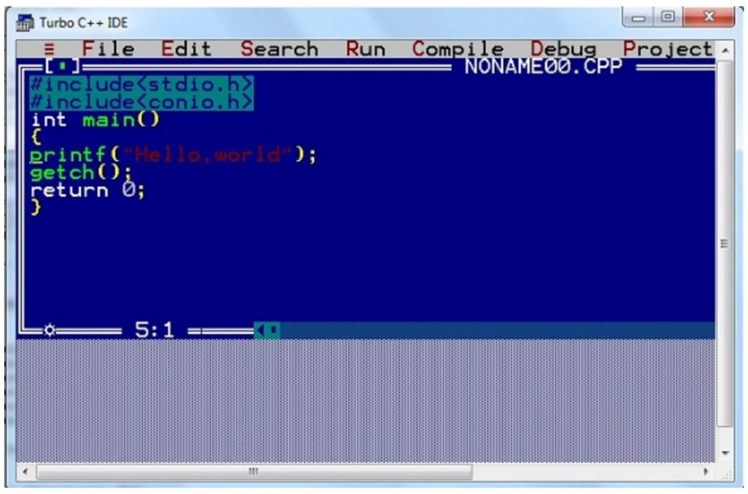


Using an IDE - Turbo C

We recommend that you use **Turbo C or Turbo C++** IDE, which is the oldest IDE for C programming. It is freely available over the Internet and is good for a beginner.

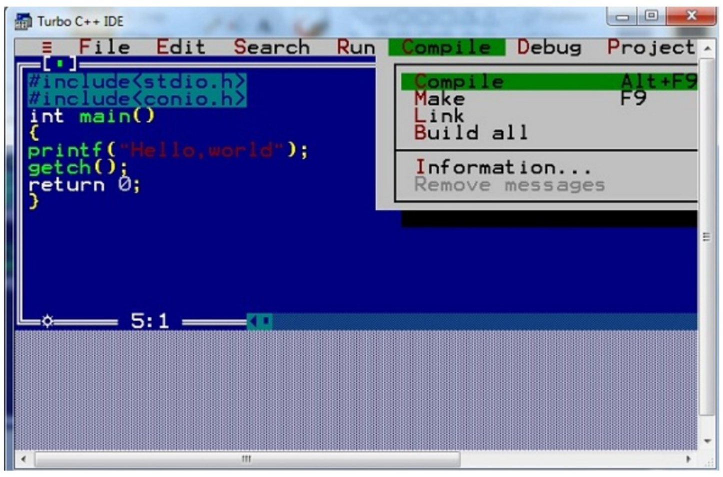
Step 1: Open Turbo C IDE (Integrated Development Environment), click on **File,** and then click on **New**

Step 2: Write a Hello World program that we created in the previous article - the **C Hello World program**.



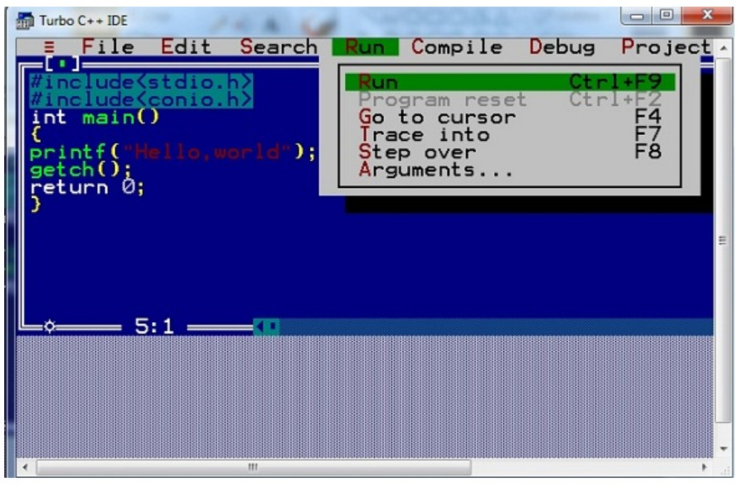
```
#include<stdio.h>
#include<conio.h>
int main()
{
printf("Hello,world");
getch();
return 0;
}
```

Step 3: Click on **Compile** menu and then on **Compile** option, or press the keys and press **Alt + F9** to compile the code.



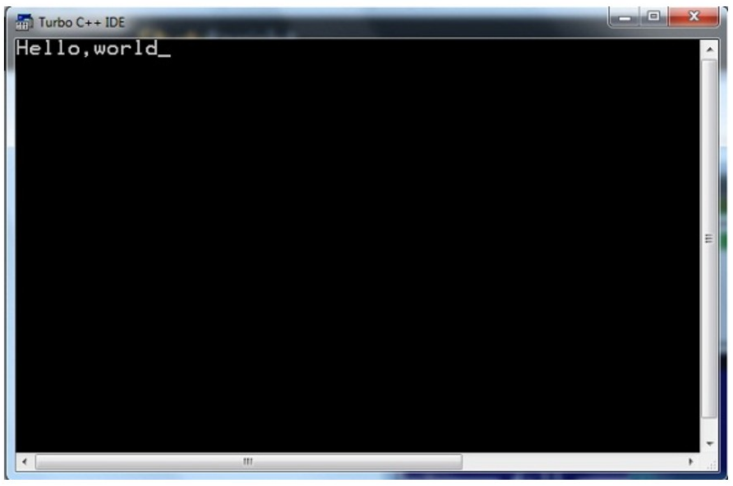
```
#include<stdio.h>
#include<conio.h>
int main()
{
printf("Hello,world");
getch();
return 0;
}
```

Step 4: Click on **Run** or press **Ctrl + F9** to run the code. Yes, C programs are first compiled to generate the object code and then that object code is Run.



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

Step 5: Output is



```
Hello,world_
```

Run C Program Without using any IDE

- If you do not wish to set up an IDE and prefer the old-school way, then download the C compiler, which is called gcc , from the GCC website https://gcc.gnu.org/install/
- Once you have downloaded and installed the gcc compiler, all you have to do is **open any text editor**, copy and paste the C program code for the C Hello World Program, and save it with the name the **helloworld.c** like any other file you save with a name.
- Now, Open the **Command prompt or Terminal**(if you use Ubuntu or Mac OS), and go to the directory where you have saved the **helloworld.c** program file.

- Type the command gcc hello.c to compile the code. This will compile the code, and if there are no errors, then it will produce an output file with the name **a.out**(default name)
- Now, to run the program, type in ./a.out , and you will see **Hello, World** displayed on your screen.

C Syntax:

Example

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

Explained:

Line 1: #include <stdio.h> is a **header file library** that lets us work with input and output functions, such as printf () (used in line 4). Header files add functionality to C programs.

Line 2: A blank line. C ignores white space. But we use it to make the code more readable.

Line 3: Another thing that always appear in a C program is main(). This is called a **function**. Any code inside its curly brackets {} will be executed.

Line 4: printf() is a **function** used to output/print text to the screen. In our example, it will output "Hello World!".

Line 5: return 0 ends the main() function.

Line 6: Do not forget to add the closing curly bracket } to actually end the main function.

Note that: Every C statement ends with a semicolon ;

Note: The body of int main() could also been written as:
int main(){printf("Hello World!");return 0;}

Remember: The compiler ignores white spaces. However, multiple lines make the code more readable.

<u>C Statements</u>

Statements

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

The following statement "instructs" the compiler to print the text "Hello World" to the screen:

<u>Example</u>

```
printf("Hello World!");
```

It is important that you end the statement with a semicolon.

If you forget the semicolon (;), an error will occur, and the program will not run:

```
error: expected ';' before 'return.'
```

Many Statements

Most C programs contain many statements.

The statements are executed, one by one, in the same order as they are written:

```
printf("Hello World!");
printf("Have a good day!");
return 0;
```

Explained

From the example above, we have three statements:

1. printf("Hello World!");
2. printf("Have a good day!");
3. return 0;

The first statement is executed first (print "Hello World!" to the screen).
Then the second statement is executed (print "Have a good day!" to the screen).
And at last, the third statement is executed (end the C program successfully).

printf Functions

You can use as many printf() functions as you want. **However**, note that it does not insert a new line at the end of the output:

Example

```
#include <stdio.h>

int main() {
  printf("Hello World!");
  printf("I am learning C.");
  printf("And it is awesome!");
  return 0;
}
```

New Lines

To insert a new line, you can use the \n character:

Example

```
#include <stdio.h>

int main()

{
  printf("Hello World!\n");
  printf("I am learning C.");
  return 0;
}
```

Tip: Two \n characters after each other will create a blank line:

Example

```
#include <stdio.h>

int main() {
  printf("Hello World!\n\n");
  printf("I am learning C.");
  return 0;
}
```

Comments in C

Comments can be used to explain code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

Comments can be **singled-lined** or **multi-lined**.

Single-line Comments

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by the compiler (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment
printf("Hello World!");
```

C Multi-line Comments

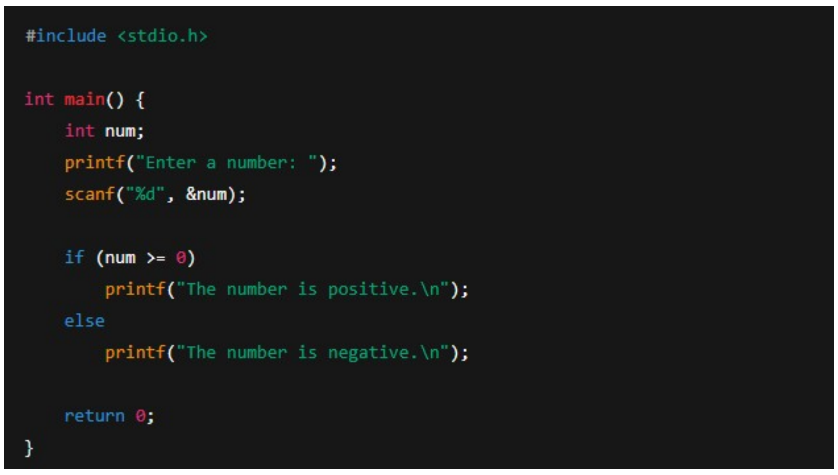
Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by the compiler:

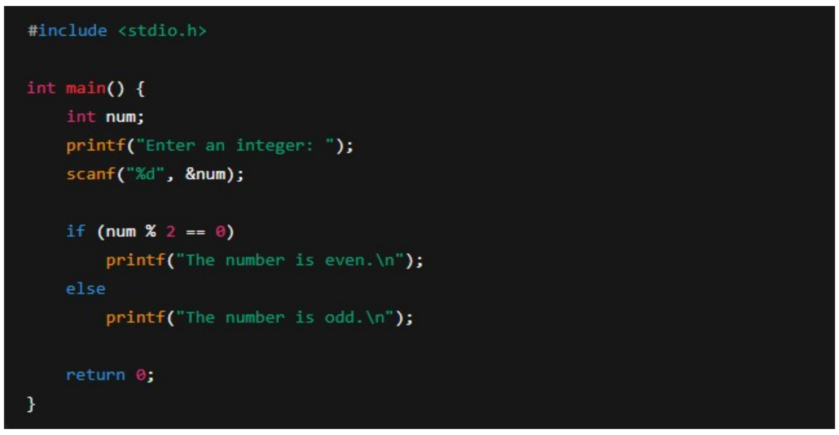
Example

```
/* The code below will print the words Hello World! to the screen,
and it is amazing */
printf("Hello World!");
```

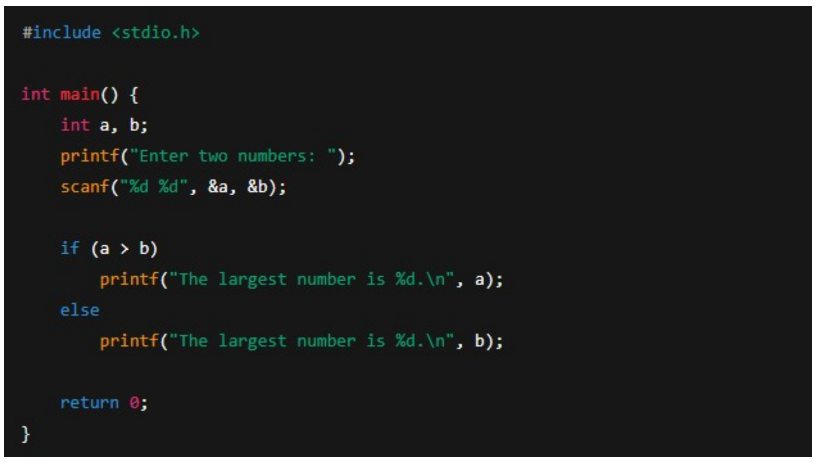
Program 1: Check if a number is positive or negative.


```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num >= 0)
        printf("The number is positive.\n");
    else
        printf("The number is negative.\n");

    return 0;
}
```

Program 2: Check if a number is even or odd.

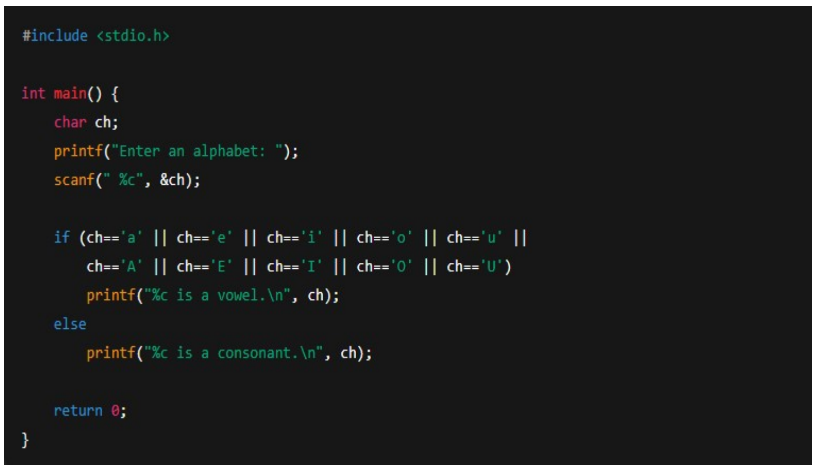

```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);

    if (num % 2 == 0)
        printf("The number is even.\n");
    else
        printf("The number is odd.\n");

    return 0;
}
```

Program 3: Find the largest of two numbers.



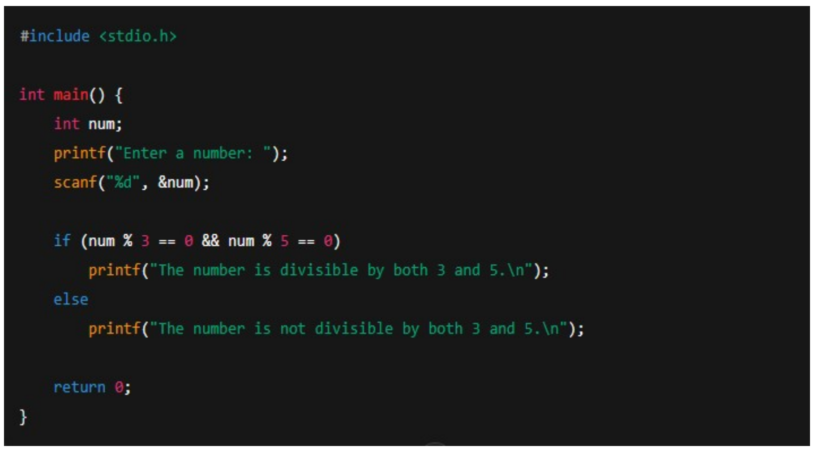
```c
#include <stdio.h>

int main() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);

    if (a > b)
        printf("The largest number is %d.\n", a);
    else
        printf("The largest number is %d.\n", b);

    return 0;
}
```

Program 4: Check if a character is a vowel or consonant.



```c
#include <stdio.h>

int main() {
    char ch;
    printf("Enter an alphabet: ");
    scanf(" %c", &ch);

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

    return 0;
}
```

Program 5: Check if a number is divisible by both 3 and 5.



```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num % 3 == 0 && num % 5 == 0)
        printf("The number is divisible by both 3 and 5.\n");
    else
        printf("The number is not divisible by both 3 and 5.\n");

    return 0;
}
```

Program 6: Simple Calculator Using Switch.

```
#include <stdio.h>
int main() {
    char op;
    float a, b;
    printf("Enter an operator (+, -, *, /): ");
    scanf(" %c", &op);
    printf("Enter two operands: ");
    scanf("%f %f", &a, &b);
    switch(op) {
        case '+':
            printf("Result = %.2f\n", a + b);
            break;
        case '-':
            printf("Result = %.2f\n", a - b);
            break;
        case '*':
            printf("Result = %.2f\n", a * b);
            break;
        case '/':
            if (b != 0)
                printf("Result = %.2f\n", a / b);
            else
                printf("Error: Division by zero\n");
            break;
        default:
            printf("Invalid operator\n");
    }
    return 0;
}
```

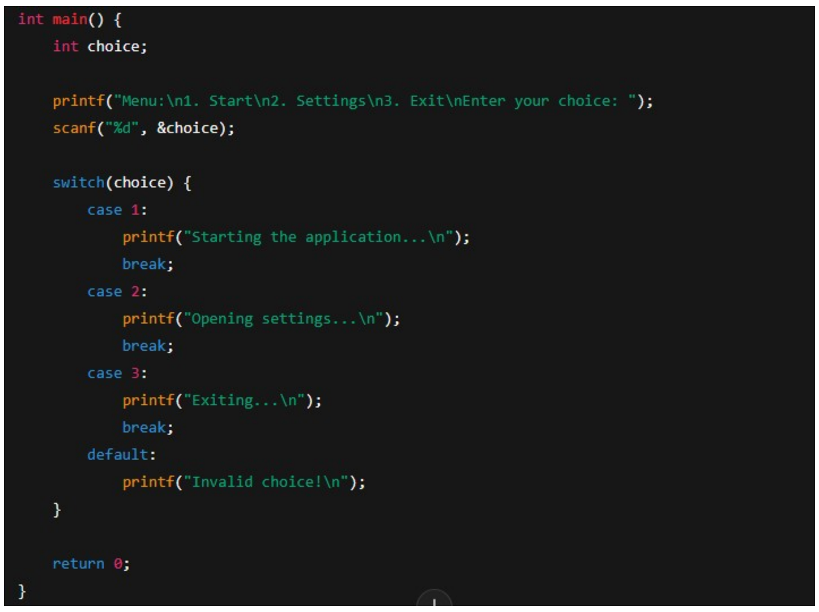
Program 7: Day of the Week Using Switch.

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

    printf("Enter day number (1 to 7): ");
    scanf("%d", &day);

    switch(day) {
        case 1:
            printf("Sunday\n");
            break;
        case 2:
            printf("Monday\n");
            break;
        case 3:
            printf("Tuesday\n");
            break;
        case 4:
            printf("Wednesday\n");
            break;
        case 5:
            printf("Thursday\n");
            break;
        case 6:
            printf("Friday\n");
            break;
        case 7:
            printf("Saturday\n");
            break;
        default:
            printf("Invalid day number\n");
    }

    return 0;
}
```

Program 8: Grade Evaluation Using Switch.

```
#include <stdio.h>

int main() {

    char grade;

    printf("Enter your grade (A/B/C/D/F): ");

    scanf(" %c", &grade);

    switch(grade) {

        case 'A':

            printf("Excellent!\n");

            break;

        case 'B':

            printf("Good job!\n");

            break;

        case 'C':

            printf("Well done\n");

            break;

        case 'D':

            printf("You passed\n");

            break;

        case 'F':
            printf("Better try again\n");
            break;
        default:
            printf("Invalid grade\n");
    }
    return 0;
}
```

Program 9: Menu-Driven Program using Switch.

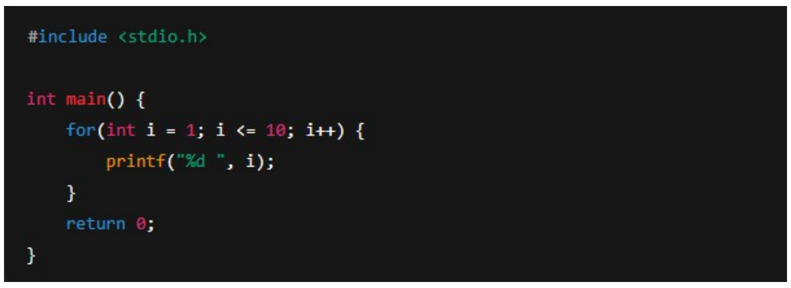


```
int main() {
    int choice;

    printf("Menu:\n1. Start\n2. Settings\n3. Exit\nEnter your choice: ");
    scanf("%d", &choice);

    switch(choice) {
        case 1:
            printf("Starting the application...\n");
            break;
        case 2:
            printf("Opening settings...\n");
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice!\n");
    }

    return 0;
}
```

Program 10: Vowel or Consonant Checker using Switch.

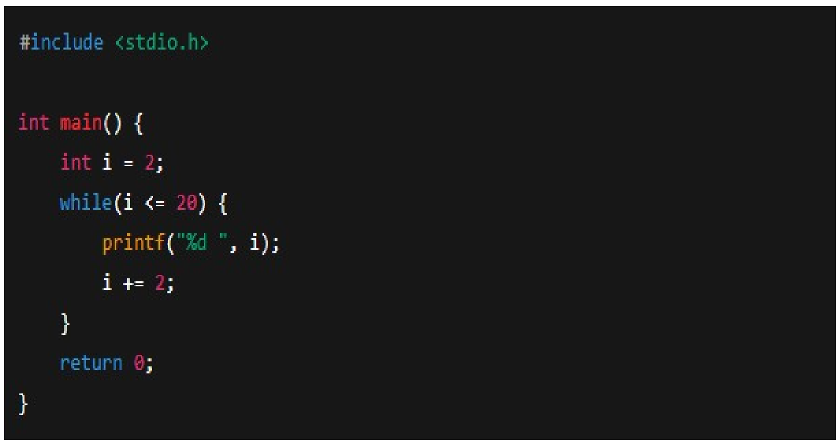
```
#include <stdio.h>
int main()
{
    char ch;

    printf("Enter a character: ");
    scanf(" %c", &ch);
    switch(ch) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':
            printf("%c is a vowel.\n", ch);
            break;
        default:
            printf("%c is a consonant.\n", ch);
    }

    return 0;
}
```

Program 11: Print Numbers from 1 to 10 using `for` loop.



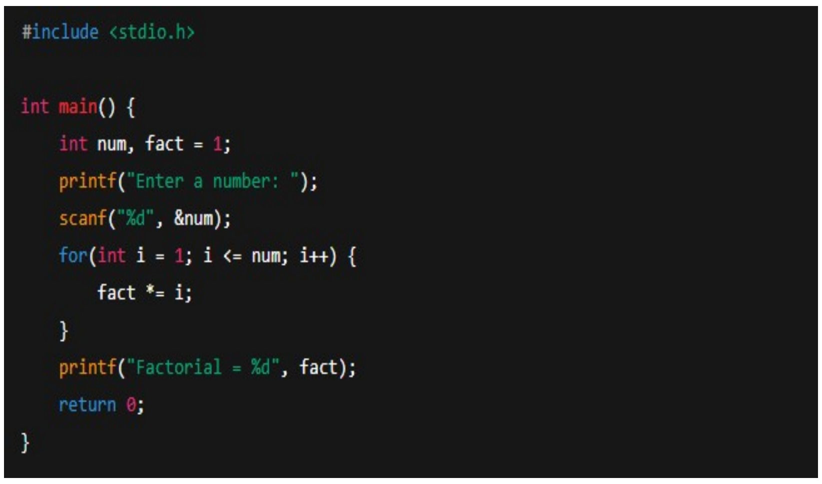
```
#include <stdio.h>

int main() {
    for(int i = 1; i <= 10; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Program 12: Print Even Numbers from 2 to 20 using `while` loop.



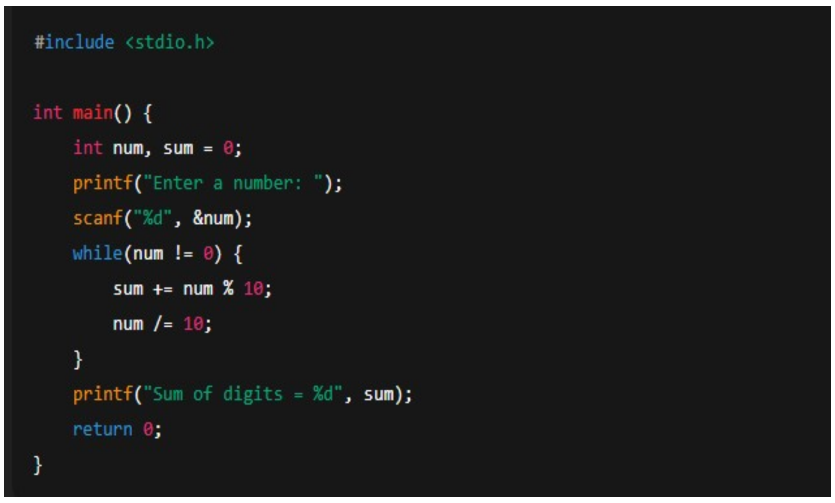
```c
#include <stdio.h>

int main() {
    int i = 2;
    while(i <= 20) {
        printf("%d ", i);
        i += 2;
    }
    return 0;
}
```

Program 13: Calculate the Factorial of a Number using for loop.



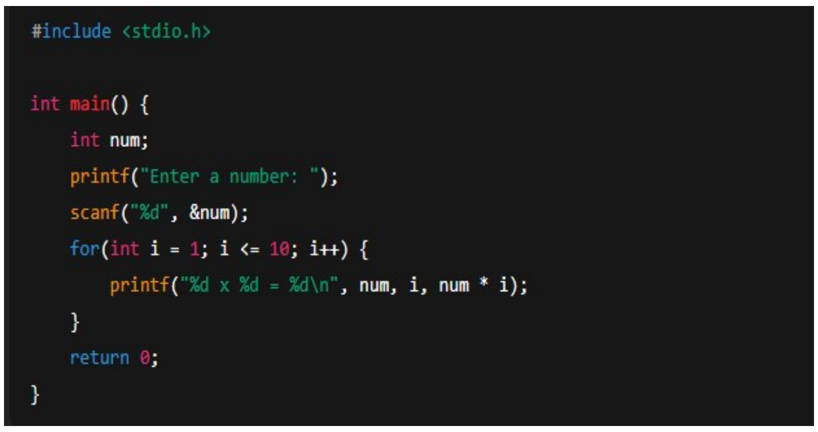
```c
#include <stdio.h>

int main() {
    int num, fact = 1;
    printf("Enter a number: ");
    scanf("%d", &num);
    for(int i = 1; i <= num; i++) {
        fact *= i;
    }
    printf("Factorial = %d", fact);
    return 0;
}
```

Program 14: Sum of Digits of a Number using while loop.



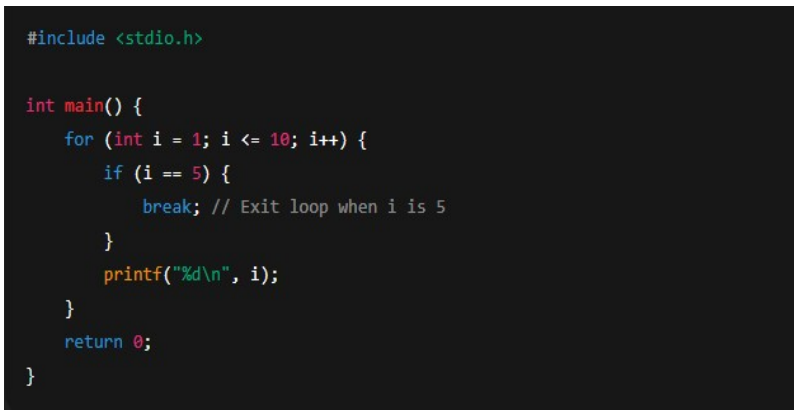
```c
#include <stdio.h>

int main() {
    int num, sum = 0;
    printf("Enter a number: ");
    scanf("%d", &num);
    while(num != 0) {
        sum += num % 10;
        num /= 10;
    }
    printf("Sum of digits = %d", sum);
    return 0;
}
```

Program 15: Print Multiplication Table of a Number using for loop**.**



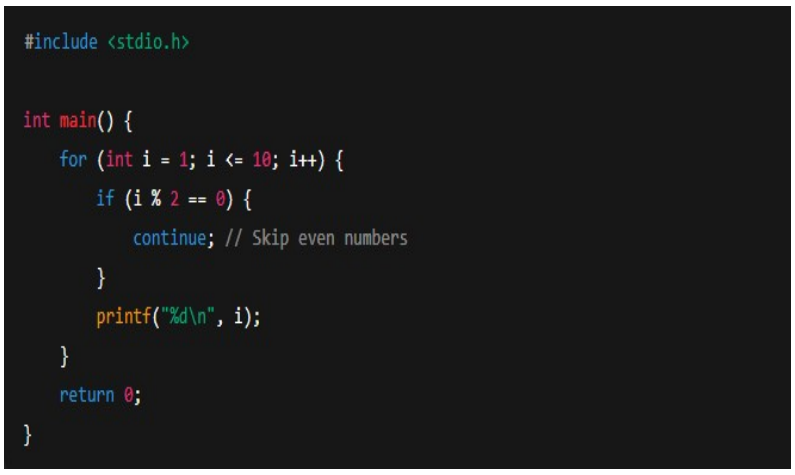
```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    for(int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", num, i, num * i);
    }
    return 0;
}
```

Program 16: Program to Print Numbers 1 to 10 but Stop at 5 using break statement.



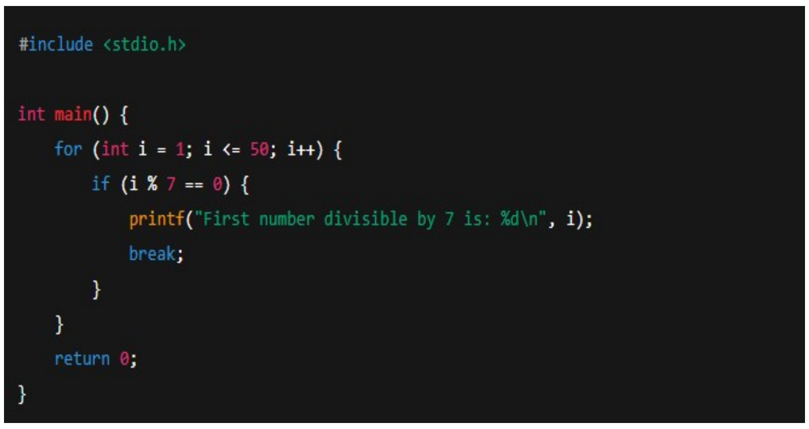
```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 5) {
            break; // Exit loop when i is 5
        }
        printf("%d\n", i);
    }
    return 0;
}
```

Program 17: Program to Print Odd Numbers Between 1 to 10 using continue statement.



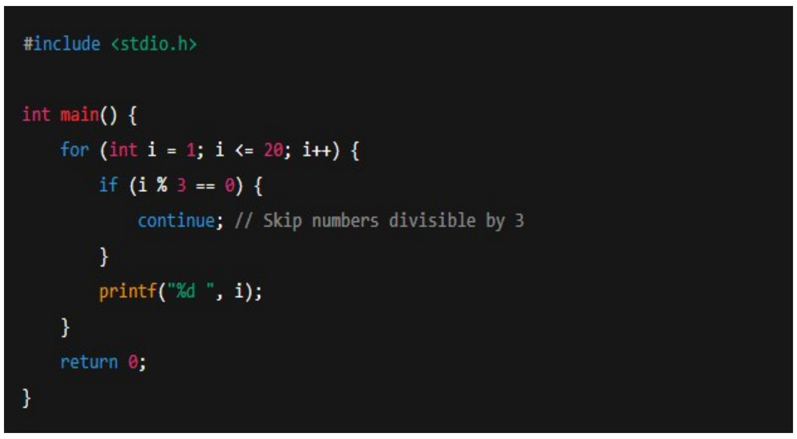
```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i % 2 == 0) {
            continue; // Skip even numbers
        }
        printf("%d\n", i);
    }
    return 0;
}
```

Program 18: Program to Find First Divisible Number by 7 in a Range using break statement.



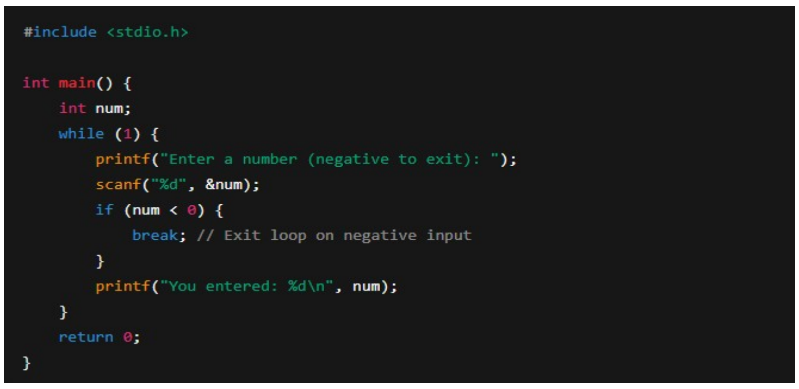
```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 50; i++) {
        if (i % 7 == 0) {
            printf("First number divisible by 7 is: %d\n", i);
            break;
        }
    }
    return 0;
}
```

Program 19: Program to Skip Multiples of 3 Between 1 to 20 using continue statement.



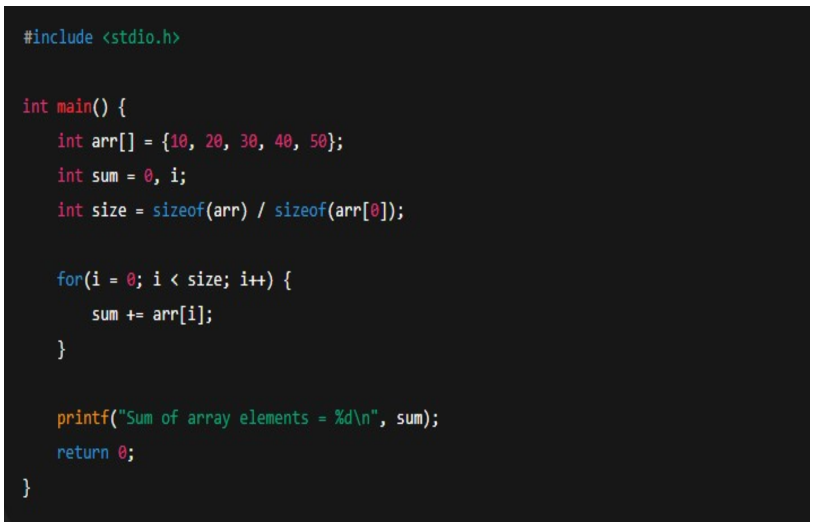
```c
#include <stdio.h>

int main() {
    for (int i = 1; i <= 20; i++) {
        if (i % 3 == 0) {
            continue; // Skip numbers divisible by 3
        }
        printf("%d ", i);
    }
    return 0;
}
```

Program 20: Program to Take Input Until Negative Number is Entered using break statement.



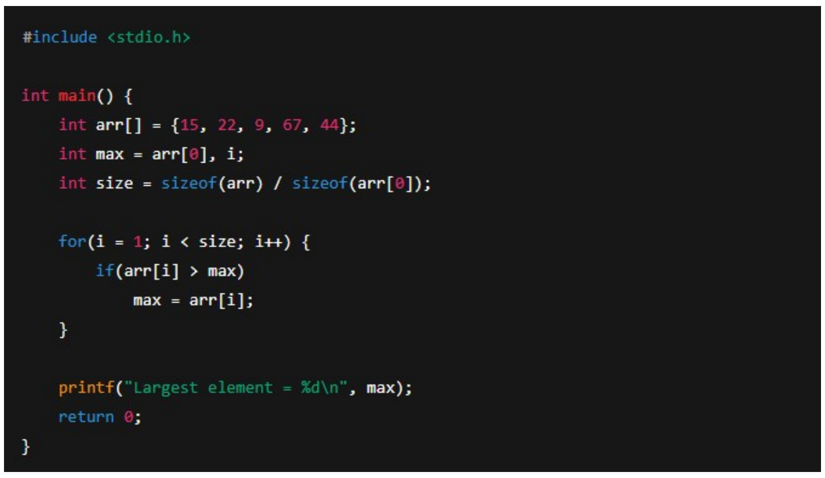
```c
#include <stdio.h>

int main() {
    int num;
    while (1) {
        printf("Enter a number (negative to exit): ");
        scanf("%d", &num);
        if (num < 0) {
            break; // Exit loop on negative input
        }
        printf("You entered: %d\n", num);
    }
    return 0;
}
```

Program 21: Program to Find the Sum of All Elements in an Array.



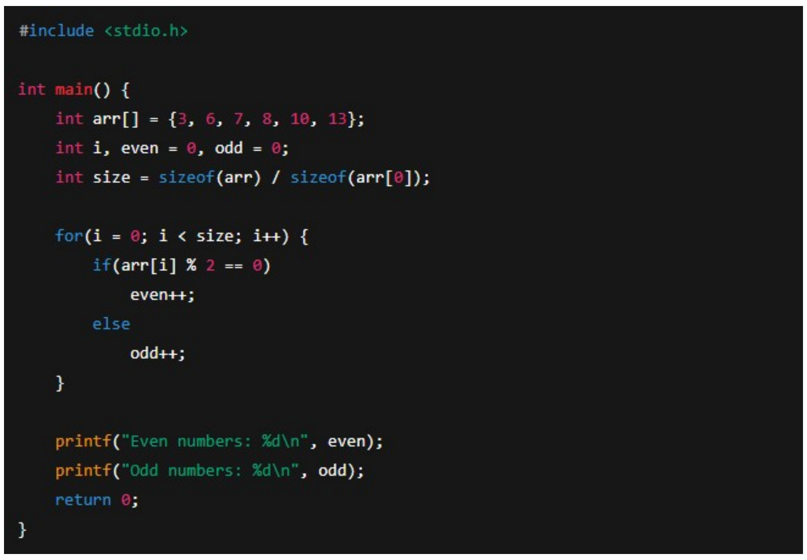
```c
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int sum = 0, i;
    int size = sizeof(arr) / sizeof(arr[0]);

    for(i = 0; i < size; i++) {
        sum += arr[i];
    }

    printf("Sum of array elements = %d\n", sum);
    return 0;
}
```

Program 22: Program to Find the Largest Element in an Array.



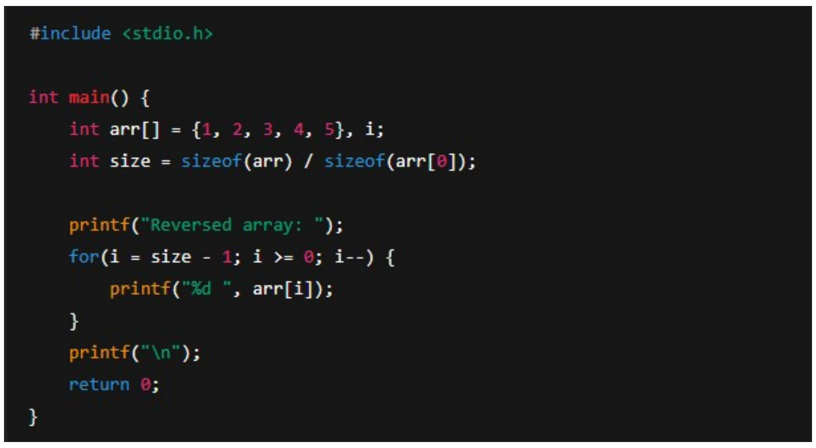
```c
#include <stdio.h>

int main() {
    int arr[] = {15, 22, 9, 67, 44};
    int max = arr[0], i;
    int size = sizeof(arr) / sizeof(arr[0]);

    for(i = 1; i < size; i++) {
        if(arr[i] > max)
            max = arr[i];
    }

    printf("Largest element = %d\n", max);
    return 0;
}
```

Program 23: Program to Count Even and Odd Elements in an Array.



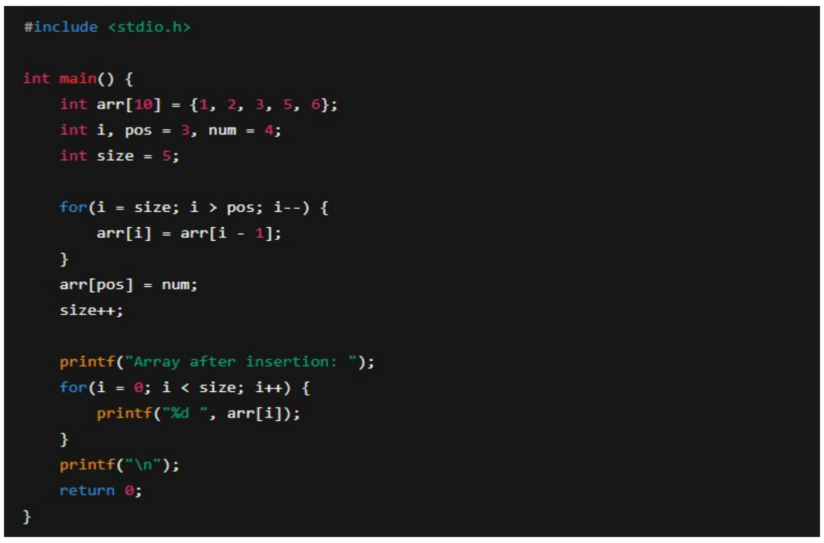
```c
#include <stdio.h>

int main() {
    int arr[] = {3, 6, 7, 8, 10, 13};
    int i, even = 0, odd = 0;
    int size = sizeof(arr) / sizeof(arr[0]);

    for(i = 0; i < size; i++) {
        if(arr[i] % 2 == 0)
            even++;
        else
            odd++;
    }

    printf("Even numbers: %d\n", even);
    printf("Odd numbers: %d\n", odd);
    return 0;
}
```

Program 24: Program to Reverse an Array.



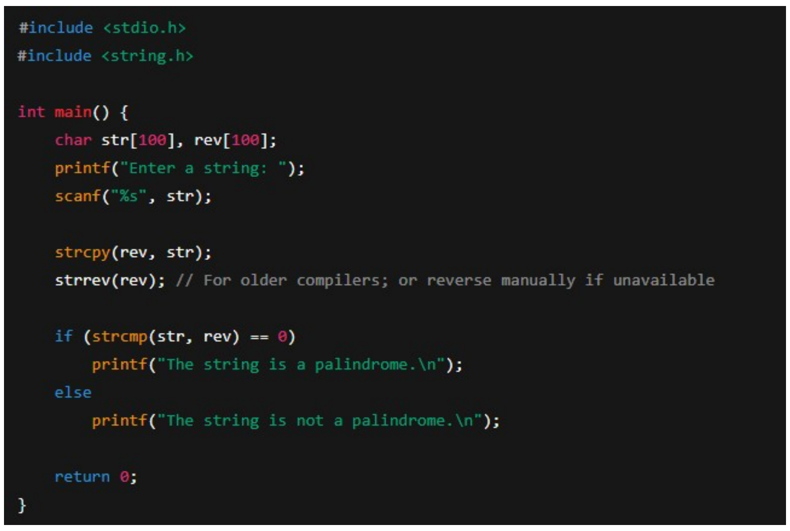
```c
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 5}, i;
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Reversed array: ");
    for(i = size - 1; i >= 0; i--) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Program 25: Program to Insert an Element in an Array.



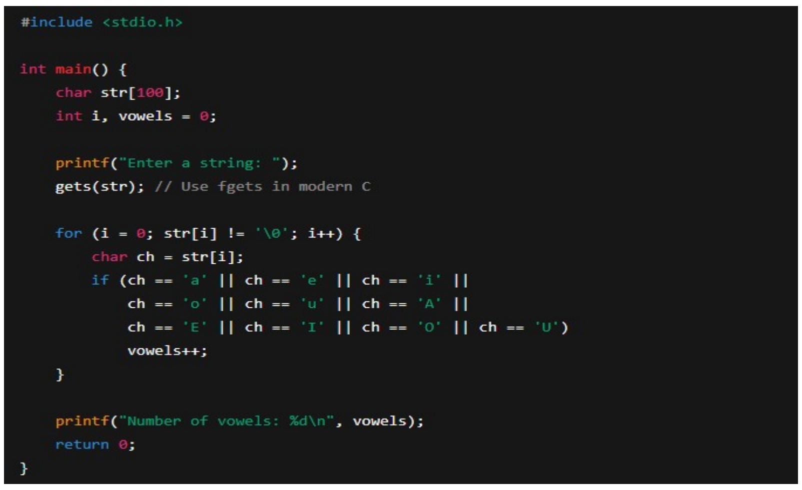
```c
#include <stdio.h>

int main() {
    int arr[10] = {1, 2, 3, 5, 6};
    int i, pos = 3, num = 4;
    int size = 5;

    for(i = size; i > pos; i--) {
        arr[i] = arr[i - 1];
    }
    arr[pos] = num;
    size++;

    printf("Array after insertion: ");
    for(i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Program 26: Check if a String is a Palindrome.



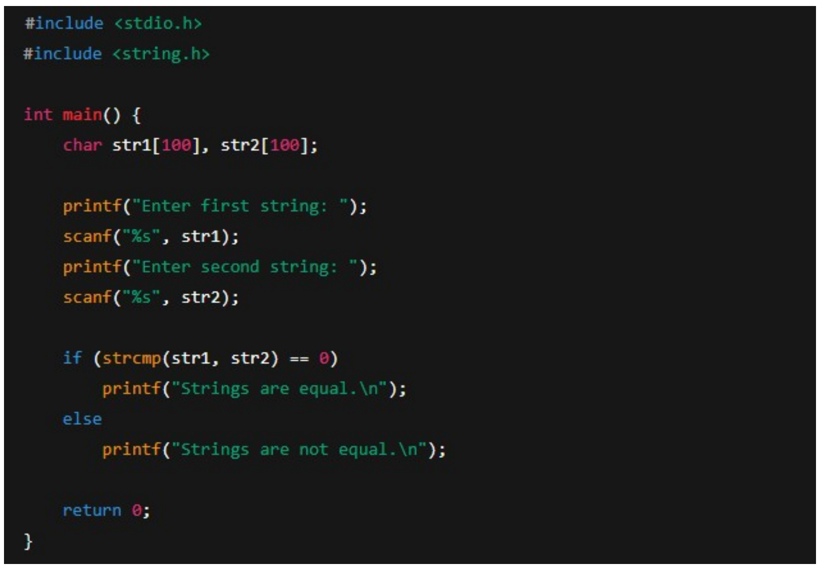
```c
#include <stdio.h>
#include <string.h>

int main() {
    char str[100], rev[100];
    printf("Enter a string: ");
    scanf("%s", str);

    strcpy(rev, str);
    strrev(rev); // For older compilers; or reverse manually if unavailable

    if (strcmp(str, rev) == 0)
        printf("The string is a palindrome.\n");
    else
        printf("The string is not a palindrome.\n");

    return 0;
}
```

Program 27: Count Vowels in a String.



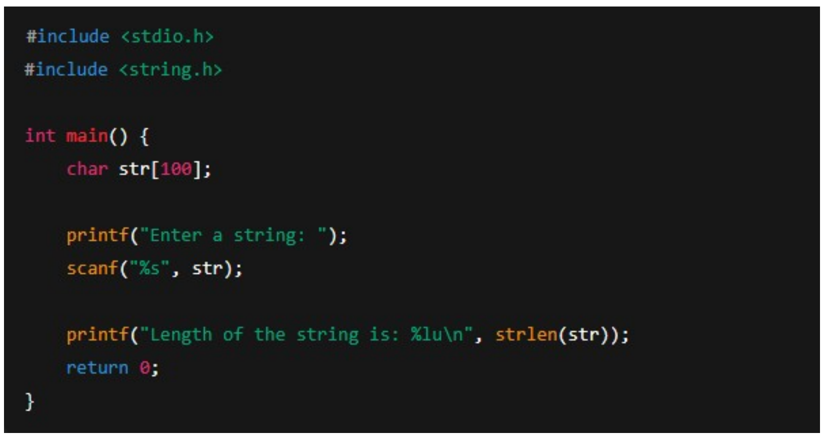
```c
#include <stdio.h>

int main() {
    char str[100];
    int i, vowels = 0;

    printf("Enter a string: ");
    gets(str); // Use fgets in modern C

    for (i = 0; str[i] != '\0'; i++) {
        char ch = str[i];
        if (ch == 'a' || ch == 'e' || ch == 'i' ||
            ch == 'o' || ch == 'u' || ch == 'A' ||
            ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U')
            vowels++;
    }

    printf("Number of vowels: %d\n", vowels);
    return 0;
}
```

Program 28: Compare Two Strings.



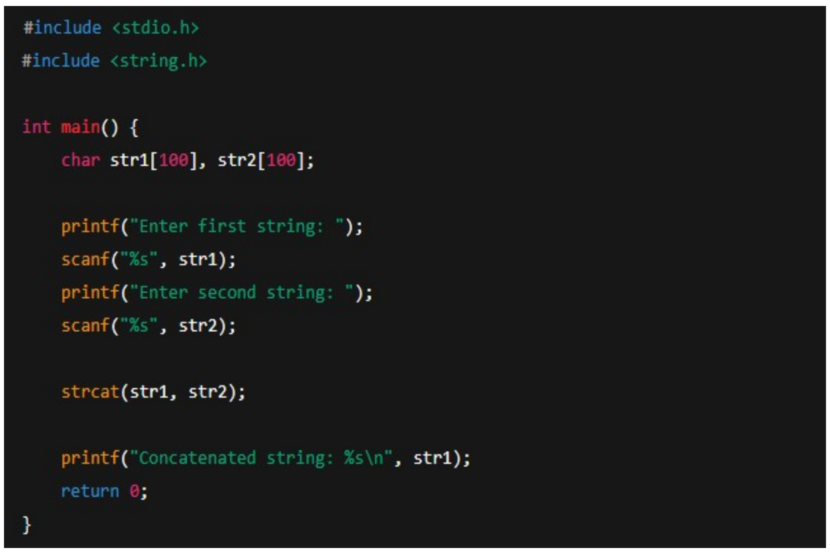
```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[100], str2[100];

    printf("Enter first string: ");
    scanf("%s", str1);
    printf("Enter second string: ");
    scanf("%s", str2);

    if (strcmp(str1, str2) == 0)
        printf("Strings are equal.\n");
    else
        printf("Strings are not equal.\n");

    return 0;
}
```

Program 29: Find Length of a String.



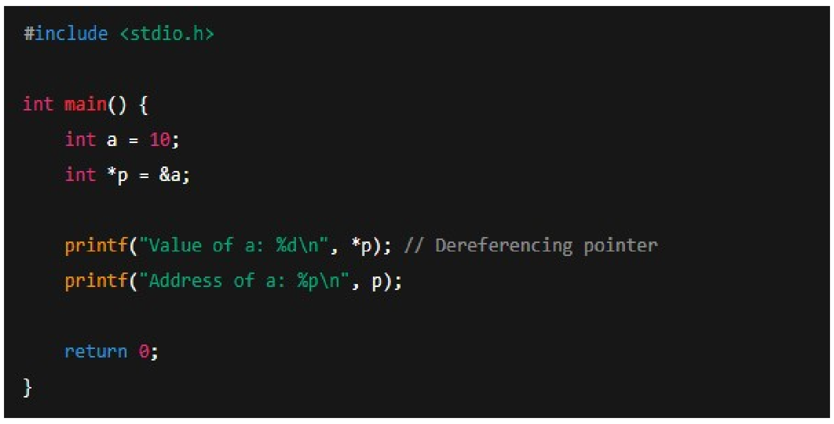
```c
#include <stdio.h>
#include <string.h>

int main() {
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str);

    printf("Length of the string is: %lu\n", strlen(str));
    return 0;
}
```

Program 30: Concatenate Two Strings.



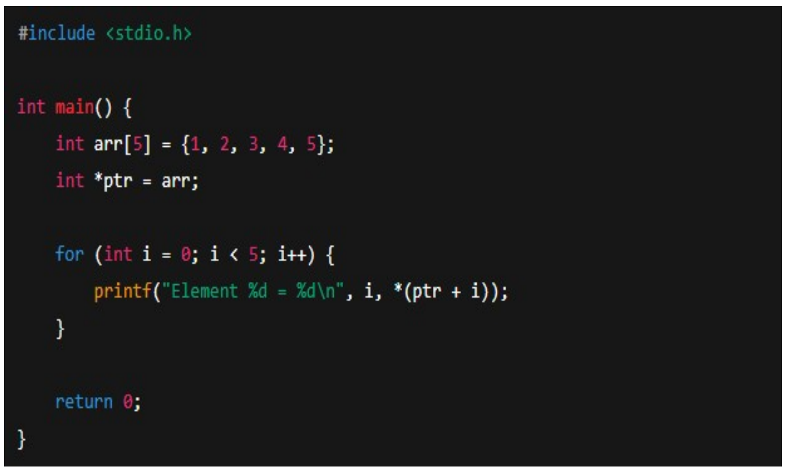
```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[100], str2[100];

    printf("Enter first string: ");
    scanf("%s", str1);
    printf("Enter second string: ");
    scanf("%s", str2);

    strcat(str1, str2);

    printf("Concatenated string: %s\n", str1);
    return 0;
}
```

Program 31: Basic Pointer Usage.

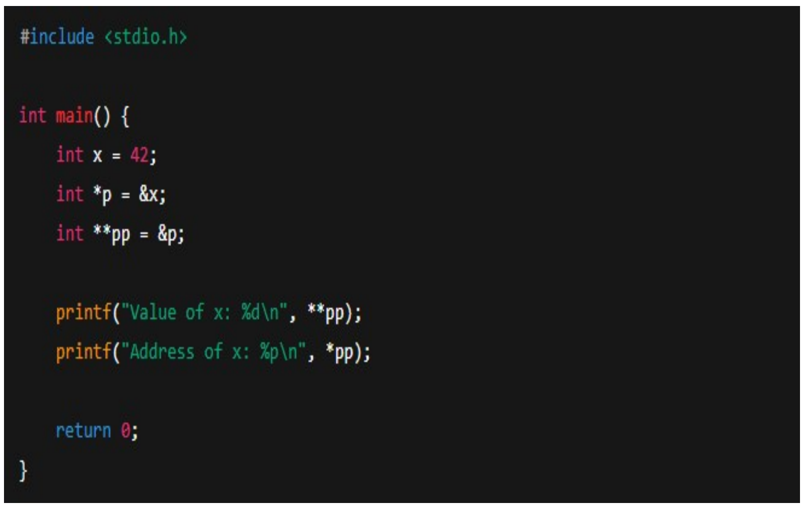


```c
#include <stdio.h>

int main() {
    int a = 10;
    int *p = &a;

    printf("Value of a: %d\n", *p); // Dereferencing pointer
    printf("Address of a: %p\n", p);

    return 0;
}
```

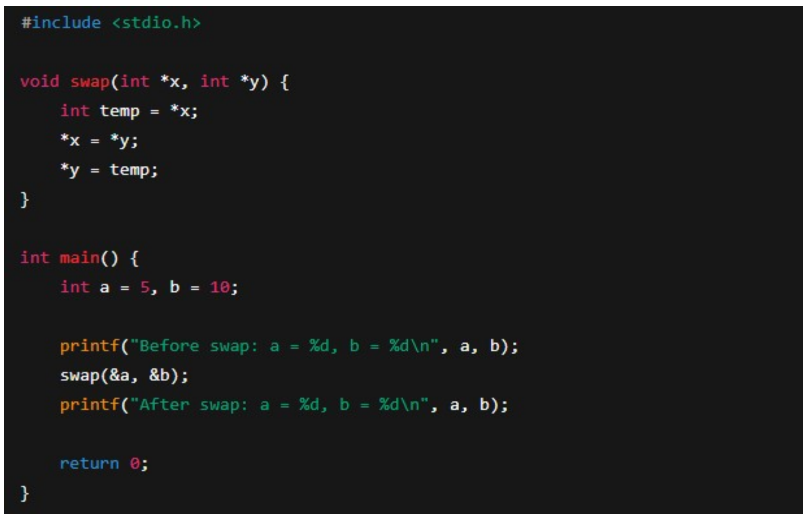
Program 32: Pointer and Array.


```c
#include <stdio.h>

int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    int *ptr = arr;

    for (int i = 0; i < 5; i++) {
        printf("Element %d = %d\n", i, *(ptr + i));
    }

    return 0;
}
```

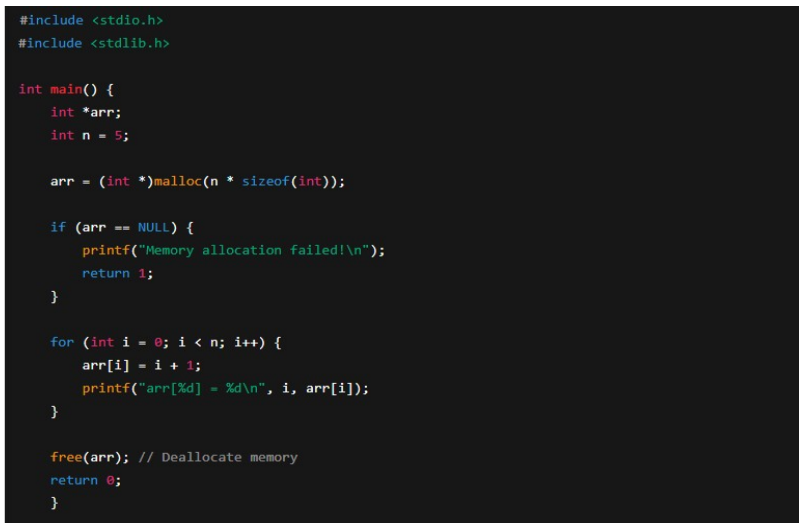
Program 33: Pointer to Pointer.


```c
#include <stdio.h>

int main() {
    int x = 42;
    int *p = &x;
    int **pp = &p;

    printf("Value of x: %d\n", **pp);
    printf("Address of x: %p\n", *pp);

    return 0;
}
```

Program 34: Swapping Values Using Pointers.


```c
#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

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

    printf("Before swap: a = %d, b = %d\n", a, b);
    swap(&a, &b);
    printf("After swap: a = %d, b = %d\n", a, b);

    return 0;
}
```

Program 35: Dynamic Memory Allocation.



```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr;
    int n = 5;

    arr = (int *)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

    for (int i = 0; i < n; i++) {
        arr[i] = i + 1;
        printf("arr[%d] = %d\n", i, arr[i]);
    }

    free(arr); // Deallocate memory
    return 0;
    }
```

LAB EXERCISE ‘C’ PROGRAM SESSION WISE

Session 1

1 –

a). Write a C program to find sum and average of three numbers.
b). Write a C program to check whether a year is a leap year or not.
c). Write a program in c to find even or odd numbers.
d). Write a C program to find the sum of individual digits of a given positive integer.
e). Write a C program to generate the first n terms of the Fibonacci sequence.

Session 2

2 –

a). Write a C program to generate prime numbers between 1 to n.
b). Write a C program to Check whether given number is Armstrong Number or Not.
c). Write a program in c to calculate power using recursion.
d). Write a program in c to function to check lowercase letter.
e). Write program to display number 1 to 10 in octal, decimal and hexadecimal system.

Session 3

3 –

a). Write a C program to generate following pattern:

```
1
1  2
1  2  3
1  2  3  4
```

b). Write a C program to generate following pattern:

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

c). Write a C program to generate following pattern:

```
1
1  1
1  1  1
1  1  1  1
```

d). Write a program to multiplication of two matrix.
e). Write a program to demonstrate multiplication table input from user till 10.