Wednesday, 19 March 2025

Assignment

QUESTION 1:

**The History and Evolution of C Programming: Its Importance and Continued Use**

* The C programming language, developed in the early 1970s by Dennis Ritchie at Bell Laboratories, has had a profound impact on the field of computer science.
* Known for its efficiency, flexibility, and portability, C remains a foundational language in modern computing.

**Origins and Development**

* C emerged as an evolution of the B programming language, which itself was derived from BCPL (Basic Combined Programming Language). Dennis Ritchie and his colleague, Ken Thompson, aimed to create a more powerful and versatile language to facilitate the development of the Unix operating system. By 1972, C had been formally developed and quickly became the language of choice for system-level programming.
* The first edition of the C programming language was documented in the book "The C Programming Language" by Brian Kernighan and Dennis Ritchie, published in 1978. This book, often referred to as "K&R C," became the definitive guide for programmers and laid the foundation for the language's standardization.

**Standardization and Evolution**

* In 1989, the American National Standards Institute (ANSI) standardized C, resulting in ANSI C, also known as C89. This standard provided consistency and compatibility across different platforms and compilers. Subsequently, the International Organization for Standardization (ISO) adopted the ANSI standard, leading to the creation of ISO C, or C90.
* Further revisions and enhancements followed, including C99 (1999), which introduced features like variable-length arrays and improved support for mathematical functions. C11 (2011) added multithreading capabilities and enhanced security features, while C17 (2017) focused on minor improvements and bug fixes. The latest standard, C23 (2023), continues to refine the language with modern features and optimizations.

**Importance and Impact**

* C's significance lies in its low-level access to system memory and hardware, making it ideal for operating system development, embedded systems, and performance-critical applications. Unix, Linux, and Windows operating systems were all heavily influenced by or written in C.
* Moreover, C has served as the foundation for many other programming languages, including C++, C#, Java, and Python. Its syntax and concepts are widely adopted, allowing programmers to transition easily between languages.

**Continued Relevance**

Despite the emergence of higher-level programming languages, C remains relevant for several reasons:

* Efficiency and Performance: C allows for direct manipulation of hardware and system resources, making it ideal for applications that require high performance.
* Portability: Programs written in C can be easily adapted to different platforms with minimal modification.
* Embedded Systems: C is extensively used in embedded systems, such as microcontrollers and IoT devices.
* Educational Value: Learning C provides a solid foundation in programming concepts, such as memory management and algorithm design.

**Conclusion**

The C programming language has stood the test of time due to its efficiency, versatility, and influence on modern computing. From operating systems to embedded systems and beyond, C continues to play a vital role in the development of technology. Its enduring relevance is a testament to its robust design and the profound impact it has had on the programming world.

QUESTION 2:

**Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated**

**Development Environment (IDE) like DevC++, VS Code, or CodeBlocks**.

**1. Install GCC Compiler**

**For Windows:**

1. Install it and choose the default options.

2. Add the MinGW bin folder (e.g., C:\MinGW\bin) to

your **Path** in system environment variables.

3. Open **Command Prompt** and type gcc --version to

check if it's installed correctly.

**For macOS:**

1. Open the **Terminal** and type xcode-select --install.
2. This installs GCC and other necessary tools.
3. Type gcc --version in the terminal to check if it’s installed.

**For Linux (Ubuntu):**

1. Open the **Terminal** and run sudo apt install build-essential.
2. Type gcc --version to verify the installation.

**2. Install and Set Up an IDE**

**For DevC++ (Windows)**:

1. Install DevC++ (it comes with MinGW).
2. Open DevC++, create a new **C Project**, and start coding.

**For Visual Studio Code (Windows, macOS, Linux)**:

1. Install the **C/C++ Extension** in VS Code from the Extensions Marketplace.

2. Ensure GCC is installed.

3. Write your code in a .c file, then press Ctrl+Shift+B to

compile and run.

**For CodeBlocks (Windows, macOS, Linux)**:

1.Choose the version with **MinGW** (for Windows).

2.Install it and open CodeBlocks.

3.Create a new **Console Application** in C, write your code, and click **Build and Run**.

QUESTION 3:

**Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.**

Headers: Include necessary libraries.

Example: #include <stdio.h>

Main Function: Where the program starts.

Example: int main() {

// code here

return 0;

}

Comments: Single-line: // This is a comment

Multi-line: /\* This is a comment \*/

Data Types: Data types in C define what kind of data a variable can store.

Example: int for integers, float for decimal numbers, char for single characters.

Variables: Store data.

Example:  
int num = 5; // Integer

float pi = 3.14; // Floating point

char letter = 'A'; // Character

Example Program:

#include <stdio.h> // Include library for I/O

int main() {

int num = 10; // Integer variable

printf("Number: %d\n", num); // Print the number

return 0; // End the program

}

EXAMPLE

#include <stdio.h>

int main() {

int age = 20; // Integer

float price = 9.99; // Floating-point number

char grade = 'A'; // Character

printf("Age: %d\n", age);

printf("Price: %.2f\n", price);

printf("Grade: %c\n", grade);

return 0;

}

QUESTION 4:

**Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.**

ARITHMETIC OPERATORS

USED FOR BASIC MATHEMATICAL OPERATIONS

**Operator** **Description Example**

+ Addition a + b

- Subtraction a - b

\* Multiplication a \* b

/ Division a / b

% Modulus (remainder) a % b

**Example:**

int a = 10, b = 5;

printf("Sum: %d\n", a + b); // Output: 15

**2. Relational (Comparison) Operators**

Used to compare values and return true (1) or false (0).

**Operator** **Description** **Example**

== Equal to a == b

!= Not equal to a != b

> Greater than a > b

< Less than a < b

>= Greater than or equal to a >= b

<= Less than or equal to a <= b

**Example:**

int a = 10, b = 5;

printf("%d\n", a > b); // Output: 1 (True)

**3. Logical Operators**

Used to combine multiple conditions.

**Operator Description Example**

&& Logical AND (a>b) && (b>0)

! Logical NOT !(a == b)

**Example:**

int a = 10, b = 5;

printf("%d\n", (a > b) && (b > 0)); // Output: 1 (True)

**4. Assignment Operators**

Used to assign values to variables.

**Operator Description Example**

= Assign value a = b

+= Add and assign a += b (a = a + b)

-= Subtract and assign a -= b

\*= Multiply and assign a \*= b

/= Divide and assign a /= b

%= Modulus and assign

a %= b

**5. Increment/Decrement Operators**

Used to increase or decrease a variable's value by 1.

**Operator Description Example**

++ Increment by 1 ++a or a++

-- Decrement by 1 --a or a--

**Example:**

int a = 5;

printf("%d\n", ++a); // Output: 6 (Pre-increment)

**6. Bitwise Operators**

Used to perform operations on individual bits.

**Operator Description Example**

& AND a & b

` ` OR

^ XOR a ^ b

~ Complement ~a

<< Left shift a << 2

>> Right shift a >> 2

**7. Conditional (Ternary) Operator**

Used as a shortcut for if-else statements.

**Operator Description Example**

? : Conditional Operator (a > b) ? a : b

**Example:**

int a = 10, b = 20;

int max = (a > b) ? a : b;

printf("Maximum: %d\n", max); // Output: 20

QUESTION 5:

**Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each**.

Decision-making statements allow a program to choose different paths based on certain conditions. These statements help control the flow of execution based on whether a condition is true or false.

In C, there are several types of decision-making statements:

* **if statement**
* **if-else statement**
* **nested if-else statement**
* **switch statement**

**1. if Statement**

The simplest form of decision-making in C is the if statement. It executes a block of code if the condition inside the if is true.

**Syntax:**

if (condition) {

// block of code to be executed if condition is true

}

**Example:**

**#include <stdio.h>**

**int main() {**

**int number = 10;**

**if (number > 0) {**

**printf("The number is positive.\n");**

**}**

**return 0;**

}

**Explanation**:

* The program checks if number is greater than 0. If true, it prints "The number is positive."

**2. if-else Statement**

The if-else statement allows you to execute one block of code if the condition is true, and another block if the condition is false.

**Syntax:**

if (condition) {

// block of code to be executed if condition is true

} else {

// block of code to be executed if condition is false

}

**Example:**

#include <stdio.h>

int main() {

int number = -5;

if (number > 0) {

printf("The number is positive.\n");

} else {

printf("The number is negative or zero.\n");

}

return 0;

}

**Explanation**:

* The program checks if number is greater than 0. If true, it prints "The number is positive." Otherwise, it prints "The number is negative or zero."

**3. nested if-else Statement**

A nested if-else statement is used when there are multiple conditions to check, and the program needs to make decisions based on a series of conditions. An if-else block can be placed inside another if or else block.

**Syntax:**

if (condition1) {

if (condition2) {

// block of code if both conditions are true

} else {

// block of code if condition1 is true and condition2 is false

}

} else {

// block of code if condition1 is false

}

**Example:**

#include <stdio.h>

int main() {

int number = 15;

if (number > 0) {

if (number % 2 == 0) {

printf("The number is positive and even.\n");

} else {

printf("The number is positive and odd.\n");

}

} else {

printf("The number is non-positive.\n");

}

return 0;

}

**Explanation**:

* The program first checks if number is positive. If true, it then checks if the number is even or odd. If number is positive and even, it prints "The number is positive and even." If it’s odd, it prints "The number is positive and odd." If the number is non-positive, it prints "The number is non-positive."

**4. switch Statement**

The switch statement is used when there are multiple possible values for a variable or expression, and the program needs to execute a different block of code for each value. It is generally used as an alternative to multiple if-else statements when you need to compare a single variable to several possible values.

**Syntax:**

switch (expression) {

case value1:

// block of code if expression == value1

break;

case value2:

// block of code if expression == value2

break;

// more cases as needed

default:

// block of code if expression does not match any case

}

**Example:**

#include <stdio.h>

int main() {

int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

case 4:

printf("Thursday\n");

break;

case 5:

printf("Friday\n");

break;

case 6:

printf("Saturday\n");

break;

case 7:

printf("Sunday\n");

break;

default:

printf("Invalid day\n");

}

return 0;

}

QUESTION 6:

**Write a C program to print numbers from 1 to 10 using all three types of loops (while, for, do-while).**

**while loop**:

* **Condition checked before** the loop starts.
* Ideal when the number of iterations is **unknown** and you need to **test** the condition **before** each iteration.
* Example: Reading input until it's valid.

while (condition) { /\* code \*/ }

**for loop**:

* Used when the number of iterations is **known** in advance.
* Best for **counting** or iterating over a range.
* Example: Printing numbers from 1 to 10.

for (int i = 0; i < 10; i++) { /\* code \*/ }

**do-while loop**:

* **Condition checked after** the loop executes, so the loop runs **at least once**.
* Ideal when the loop body must execute **at least once**, regardless of the condition.
* Example: Prompting for user input until it's correct.

do { /\* code \*/ } while (condition);

QUESTION 7:

**Explain the use of break, continue, and goto statements in C. Provide examples of each**

**1. Break Statement**

The break statement is used to **exit** from a loop or a switch statement prematurely, regardless of the loop's or switch statement's condition.

EXAMPLE:

#include <stdio.h>

int main() {

for (int i = 1; i <= 10; i++) {

if (i == 5) {

break; // Exit the loop when i equals 5

}

printf("%d ", i); // This will print numbers from 1 to 4

}

return 0;

}

**2. Continue Statement**

The continue statement is used to **skip the current iteration** of a loop and **move to the next iteration**. The remaining code inside the loop for the current iteration is skipped, and the loop proceeds with the next iteration (if the condition is still true).

EXAMPLE:

#include <stdio.h>

int main() {

for (int i = 1; i <= 10; i++) {

if (i % 2 == 0) {

continue; // Skip even numbers

}

printf("%d ", i); // This will print only odd numbers (1, 3, 5, 7, 9)

}

return 0;

}

**3. Goto Statement**

The goto statement is used to **jump to another part of the program**, typically identified by a label. It is often considered **bad practice** in structured programming because it can make the flow of execution harder to follow and understand.

EXAMPLE:

#include <stdio.h>

int main() {

int i = 0;

start: // Label

if (i >= 5) {

return 0; // Exit if i is 5 or more

}

printf("%d ", i);

i++;

goto start; // Jump back to the "start" label

return 0;

}

QUESTION 8:

**Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples**.

**Concept of Arrays in C**

An **array** in C is a collection of elements of the same type stored in contiguous memory locations. The elements in an array are accessed using an index, starting from 0. Arrays allow you to store and manage multiple values efficiently under a single variable name.

**Syntax for declaring an array:**

type array\_name[size];

* type: The data type of elements in the array (e.g., int, float, etc.).
* array\_name: The name of the array.
* size: The number of elements in the array.

**One-Dimensional Array**

A **one-dimensional array** is a list of elements of the same type. It is like a row of data, where each element is accessed using a single index.

EXAMPLE:

#include <stdio.h>

int main() {

int arr[5] = {1, 2, 3, 4, 5}; // Declaring and initializing a 1D array

// Printing elements of the array

for (int i = 0; i < 5; i++) {

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

}

return 0;

}

**Multi-Dimensional Arrays**

A **multi-dimensional array** is an array of arrays. It can have two or more dimensions (e.g., 2D, 3D arrays). The most common multi-dimensional array is the **two-dimensional array**, which can be visualized as a matrix (rows and columns).

**Syntax for a two-dimensional array:**

type array\_name[rows][columns];

EXAMPLE:

#include <stdio.h>

int main() {

int arr[3][3] = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

}; // Declaring and initializing a 2D array (3x3 matrix)

// Printing elements of the 2D array

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

printf("%d ", arr[i][j]);

}

printf("\n"); // New line after each row

}

return 0;

}

QUESTION 9:

**Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.**

**1. strlen()**

The strlen() function is used to determine the length of a string (excluding the null terminator \0).

EXAMPLE:

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Hello, World!";

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

return 0;

}

**2. strcpy()**

The strcpy() function is used to copy one string into another. It copies the contents of the source string (including the null terminator) to the destination string.

EXAMPLE

#include <stdio.h>

#include <string.h>

int main() {

char src[] = "Hello";

char dest[20];

strcpy(dest, src); // Copying 'src' into 'dest'

printf("Source: %s\n", src);

printf("Destination: %s\n", dest);

return 0;

}

**3. strcat()**

The strcat() function is used to concatenate (join) two strings. It appends the source string to the destination string.

EXAMPLE:

#include <stdio.h>

#include <string.h>

int main() {

char str1[50] = "Hello";

char str2[] = " World!";

strcat(str1, str2); // Concatenate 'str2' to 'str1'

printf("Concatenated string: %s\n", str1);

return 0;

}

**4. strcmp()**

The strcmp() function is used to compare two strings. It compares the strings lexicographically (based on their ASCII values).

EXAMPLE:

#include <stdio.h>

#include <string.h>

int main() {

char str1[] = "Hello";

char str2[] = "World";

int result = strcmp(str1, str2);

if (result == 0) {

printf("The strings are equal.\n");

} else if (result < 0) {

printf("'%s' is less than '%s'.\n", str1, str2);

} else {

printf("'%s' is greater than '%s'.\n", str1, str2);

}

return 0;

}

**5. strchr()**

The strchr() function is used to find the first occurrence of a character in a string.

EXAMPLE:

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Hello, World!";

char \*result = strchr(str, 'o'); // Find first occurrence of 'o'

if (result != NULL) {

printf("First occurrence of 'o': %s\n", result);

} else {

printf("'o' not found.\n");

}

return 0;

}