1. What is the purpose of the main function in a C++ program?

- The main function is the entry point for execution in a C++ program. It is where the program starts running.

2. Explain the significance of the return type of the main function.

- The return type of the main function is typically int, indicating that it will return an integer value to the operating system when the program ends. This is often used to indicate whether the program executed successfully (return 0; for success, and non-zero for errors).

3. What are the two valid signatures of the main function in C++?

- int main() – Main function with no arguments.

- int main(int argc, char *argv[]) – Main function that takes command-line arguments, where argc is the count of arguments and argv is the array of arguments.

4. What is function prototyping and why is it necessary in C++?

- A function prototype provides a declaration of a function before its actual definition. It specifies the function's return type, name, and parameters, and helps the compiler know how to call the function correctly before its actual implementation is encountered in the code.

5. How do you declare a function prototype for a function that returns an integer and takes two integer parameters?

- int functionName(int, int);

6. What happens if a function is used before it is prototyped?

- If a function is called before its prototype or definition, the compiler will generate an error because it doesn't know how to call the function correctly.

Certainly! Let's continue addressing the remaining questions from your assignment:

---

7. What is the difference between a declaration and a definition of a function?

- **Declaration**: Specifies the function's name, return type, and parameters, but does not provide the implementation. It tells the compiler about the function's existence.

- **Definition**: Provides the actual implementation of the function, including the body. It allocates memory for the function.

Example:

// Declaration

int add(int, int);

// Definition

```cpp
int add(int a, int b) {

    return a + b;

}
```

---

8. How do you call a simple function that takes no parameters and returns void?

```cpp
void greet() {

    std::cout << "Hello, World!";

}


int main() {

    greet();  // Function call

    return 0;

}
```

---

9. Explain the concept of "scope" in the context of functions.

Scope refers to the region of the program where a variable or function is accessible.

- **Local Scope**: Variables declared inside a function are accessible only within that function.
- **Global Scope**: Variables declared outside all functions are accessible throughout the program.

```cpp
int globalVar = 10;  // Global scope


void func() {

    int localVar = 20;  // Local scope

    std::cout << globalVar << " " << localVar;

}
```

---

10. What is call by reference in C++?

In call by reference, a function receives references to the actual variables passed, allowing it to modify the original variables.

```cpp
void increment(int &x) {

    x++;

}
```

```cpp
int main() {

    int num = 5;

    increment(num);

    std::cout << num;  // Outputs 6

    return 0;

}
```

---

11. How does call by reference differ from call by value?

- **Call by Value**: The function receives a copy of the argument's value; changes made to the parameter do not affect the original argument.

- **Call by Reference**: The function receives a reference to the argument; changes made to the parameter affect the original argument.

---

12. Provide an example of a function that uses call by reference to swap two integers.

```cpp
void swap(int &a, int &b) {

    int temp = a;

    a = b;

    b = temp;

}


int main() {

    int x = 3, y = 4;

    swap(x, y);

    std::cout << x << " " << y;  // Outputs 4 3

    return 0;

}
```

---

13. What is an inline function in C++?

An inline function is a function where the compiler attempts to insert the function's code directly at the point of call, rather than performing a traditional function call. This can reduce overhead for small, frequently called functions.(GeeksforGeeks)

```
inline int square(int x) {

    return x * x;

}
```

---

14. How do inline functions improve performance?

By eliminating the overhead of function calls, inline functions can improve performance, especially for small functions that are called frequently. However, excessive inlining can increase the size of the binary and may have the opposite effect.(Cplusplus.com)

---

15. Explain the syntax for declaring an inline function.

```
inline return_type function_name(parameters) {

    // function body

}
```

For example:

```
inline int add(int a, int b) {

    return a + b;

}
```

---

16. What are macros in C++ and how are they different from inline functions?

- **Macros**: Preprocessor directives that define code snippets to be inserted at compile time. They are not type-checked and can lead to errors if not used carefully.

- **Inline Functions**: Functions defined with the inline keyword that are type-checked and can be debugged. They are generally safer and more flexible than macros.

---

17. Explain the advantages and disadvantages of using macros over inline functions.

Advantages of Macros:

- Can be used for code snippets that are not functions.

- May offer slightly faster execution since they are expanded at compile time.

Disadvantages of Macros:

- Lack type safety.

- Cannot be debugged easily.

- Can lead to code bloat and maintenance issues.

Advantages of Inline Functions:

- Type-safe and can be debugged.(C++ Reference)

- Can access class members and have scope.(GeeksforGeeks)

Disadvantages of Inline Functions:

- May increase binary size if overused.(GeeksforGeeks)

- Not suitable for large functions.(GeeksForGeeks)

---

18. Provide an example to illustrate the differences between macros and inline functions.

Macro Example:

#define SQUARE(x) ((x) * (x))

Inline Function Example:

inline int square(int x) {

   return x * x;

}

Key Differences:

- Macros are expanded by the preprocessor and lack type safety.(GeeksforGeeks)

- Inline functions are type-checked and can be debugged.

---

19. What is function overloading in C++?

Function overloading allows multiple functions to have the same name but differ in the number or type of their parameters. The compiler differentiates them based on the function signature.(GeeksforGeeks)

void print(int i) {

   std::cout << "Integer: " << i;

}

void print(double d) {

   std::cout << "Double: " << d;

}

Certainly! Let's continue addressing the remaining questions from your assignment:

---

20. How does the compiler differentiate between overloaded functions?

The compiler differentiates overloaded functions based on their **function signatures**, which include:

- **Number of parameters**: Functions with different numbers of parameters are considered distinct.(Learn C++)

- **Type of parameters**: Functions with parameters of different types are considered distinct.

Note: The return type is **not** considered part of the function signature for overloading purposes.(Learn C++)

Example:

int add(int x, int y) { return x + y; }   // add(int, int)

double add(double x, double y) { return x + y; } // add(double, double)

In this example, the compiler can distinguish between the two add functions based on the parameter types.

---

21. Provide an example of overloaded functions in C++.

```
#include <iostream>

void print(int i) {
   std::cout << "Integer: " << i << std::endl;
}

void print(double d) {
   std::cout << "Double: " << d << std::endl;
}

int main() {
   print(5);     // Calls print(int)
   print(3.14);   // Calls print(double)
   return 0;
}
```

In this example, the print function is overloaded to handle both integers and doubles.

---

22. What are default arguments in C++?

Default arguments in C++ are values provided for function parameters in the function declaration. If a function is called without providing a value for those parameters, the default values are used.(GeeksforGeeks)

Example:

#include <iostream>

void greet(std::string name = "Guest") {

   std::cout << "Hello, " << name << "!" << std::endl;

}

int main() {

   greet();   // Uses default argument

   greet("Alice");   // Uses provided argument

   return 0;

}

In this example, the greet function has a default argument for the name parameter.

---

23. How do you specify default arguments in a function declaration?

Default arguments are specified in the function declaration (or prototype), not in the definition.(GeeksforGeeks)

Example:

void display(int x = 0, int y = 0); // Declaration with default arguments

void display(int x, int y) {   // Definition

   std::cout << "x: " << x << ", y: " << y << std::endl;

}

In this example, the default values for x and y are specified in the declaration.(GeeksforGeeks)

---

24. What are the rules for using default arguments in functions?

1. **Specify default values in the declaration**: Default values should be provided in the function declaration, not in the definition.(GeeksforGeeks)

2. **Provide default values from right to left**: If a function has multiple parameters, default values must be provided from the rightmost parameter to the left.(GeeksforGeeks)

3. **Avoid ambiguity**: Overloading functions with default arguments can lead to ambiguity if the compiler cannot determine which function to call.(GeeksforGeeks)

Example:

void func(int x, int y = 10);   // Valid

void func(int x = 5, int y);   // Invalid: default argument for 'x' comes before 'y'

---

25. Provide an example of a function with default arguments.

#include <iostream>

void display(int x = 1, int y = 2) {

   std::cout << "x: " << x << ", y: " << y << std::endl;

}

int main() {

   display();   // Uses default arguments

   display(10);   // Uses default y

   display(10, 20);   // Uses provided arguments

   return 0;

}

In this example, the display function has default arguments for both x and y.

---

26. Write a C++ program that prints "Hello, World!" using the main function.

#include <iostream>

int main() {

   std::cout << "Hello, World!" << std::endl;

   return 0;

}

This program prints "Hello, World!" to the console.

---

27. Create a program that takes two integers as command line arguments and prints their sum.

```cpp
#include <iostream>
#include <cstdlib> // For atoi()

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <num1> <num2>" << std::endl;
        return 1;
    }
    int num1 = std::atoi(argv[1]);
    int num2 = std::atoi(argv[2]);
    std::cout << "Sum: " << num1 + num2 << std::endl;
    return 0;
}
```

This program reads two integers from the command line and prints their sum.

---

28. Write a function prototype for a function that calculates the factorial of a number.

```cpp
long long factorial(int n);
```

This is the prototype for a function that calculates the factorial of a number.

---

29. Implement a C++ program with a function prototype for a function that returns the maximum of three numbers.

```cpp
#include <iostream>

int maxOfThree(int a, int b, int c);

int main() {
    int a = 5, b = 10, c = 3;
    std::cout << "Maximum: " << maxOfThree(a, b, c) << std::endl;
    return 0;
}
```

```cpp
int maxOfThree(int a, int b, int c) {

    return (a > b) ? (a > c ? a : c) : (b > c ? b : c);

}
```

This program defines a function `max

Absolutely! Let's continue addressing the remaining questions from your assignment:

---

36. Write an inline function that calculates the square of a number.

Inline functions are defined using the inline keyword, suggesting to the compiler that the function's code should be inserted directly at the point of call, thereby eliminating the overhead of a function call.(GeeksforGeeks)

Example:

```cpp
#include <iostream>


inline int square(int x) {

    return x * x;

}


int main() {

    int num = 5;

    std::cout << "Square of " << num << " is: " << square(num) << std::endl;

    return 0;

}
```

In this example, the square function is declared as inline, and it calculates the square of the given number.

---

37. Create an inline function that returns the cube of a number.

Similarly, we can define an inline function to calculate the cube of a number.

Example:

```cpp
#include <iostream>


inline int cube(int x) {

    return x * x * x;
```

```
}
```

```
int main() {

    int num = 4;

    std::cout << "Cube of " << num << " is: " << cube(num) << std::endl;

    return 0;

}
```

In this example, the cube function is defined as inline, and it calculates the cube of the given number.

---

38. Implement a program that uses an inline function to calculate the sum of two numbers.

By using an inline function, we can directly insert the function's code at the point of call, making the addition operation more efficient.(O'Reilly Media)

Example:

```
#include <iostream>
```

```
inline int add(int a, int b) {

    return a + b;

}
```

```
int main() {

    int num1 = 10, num2 = 20;

    std::cout << "Sum of " << num1 << " and " << num2 << " is: " << add(num1, num2) << std::endl;

    return 0;

}
```

In this example, the add function is declared as inline, and it calculates the sum of the two given numbers.

---

39. Write a macro to calculate the square of a number and compare its performance with an inline function.

Macros are preprocessor directives that perform textual substitution before compilation. They can be used to define simple operations like squaring a number.(GeeksforGeeks)

Macro Definition:

#define SQUARE(x) ((x) * (x))

Comparison:

- **Inline Function:**

    o   Type-safe: Ensures that the argument is evaluated once.

    o   Can be debugged like regular functions.(GeeksforGeeks)

    o   Scope and type checking are enforced.(GeeksforGeeks)

- **Macro:**

    o   No type safety: Arguments are evaluated multiple times, which can lead to unexpected results.(GeeksforGeeks)

    o   Cannot be debugged easily.(Wikipedia)

    o   No scope or type checking.(GeeksforGeeks)

Performance Consideration:

While both macros and inline functions can eliminate the overhead of a function call, inline functions provide better safety and debugging capabilities. Macros, being simple text substitutions, can lead to unexpected behaviors if not used carefully.

---

40. Implement a macro to find the maximum of two numbers and compare it with an inline function.

We can define a macro to find the maximum of two numbers.

Macro Definition:

#define MAX(a, b) ((a) > (b) ? (a) : (b))

Inline Function Definition:

inline int max(int a, int b) {

   return (a > b) ? a : b;

}

Comparison:

- **Inline Function:**

    o   Type-safe: Arguments are evaluated once.

    o   Can be debugged like regular functions.

    o   Scope and type checking are enforced.

- **Macro:**

    o   No type safety: Arguments are evaluated multiple times, which can lead to unexpected results.(GeeksforGeeks)

        o   Cannot be debugged easily.([GeeksforGeeks](#))

        o   No scope or type checking.([GeeksforGeeks](#))

Performance Consideration:

Inline functions are generally preferred over macros due to their type safety and debugging capabilities. Macros, while potentially faster due to direct substitution, can introduce subtle bugs if not used carefully.

Certainly! Let's continue addressing the remaining questions from your assignment:

---

41. Write overloaded functions to calculate the area of a circle, rectangle, and triangle.

Function overloading allows multiple functions with the same name but different parameter types or numbers.([Learn C++](#))

Example:

#include <iostream>

#include <cmath> // For M_PI

// Overloaded function to calculate area of a circle

double area(double radius) {

   return M_PI * radius * radius;

}

// Overloaded function to calculate area of a rectangle

double area(double length, double width) {

   return length * width;

}

// Overloaded function to calculate area of a triangle

double area(double base, double height, bool) { // 'bool' is a dummy parameter to differentiate

   return 0.5 * base * height;

}

int main() {

```
    std::cout << "Area of circle: " << area(5.0) << std::endl;

    std::cout << "Area of rectangle: " << area(4.0, 6.0) << std::endl;

    std::cout << "Area of triangle: " << area(4.0, 6.0, true) << std::endl;

    return 0;

}
```

In this example, the area function is overloaded to calculate the area of a circle, rectangle, and triangle based on the number and type of parameters passed.

---

42. Implement overloaded functions to find the maximum of two and three numbers.

Function overloading can be used to create multiple functions with the same name but different parameter lists.

Example:

```
#include <iostream>


// Overloaded function to find maximum of two numbers

int max(int a, int b) {

    return (a > b) ? a : b;

}


// Overloaded function to find maximum of three numbers

int max(int a, int b, int c) {

    return max(max(a, b), c);

}


int main() {

    std::cout << "Maximum of 3 and 5: " << max(3, 5) << std::endl;

    std::cout << "Maximum of 3, 5, and 7: " << max(3, 5, 7) << std::endl;

    return 0;

}
```

In this example, the max function is overloaded to find the maximum of two and three integers.

---

43. Create overloaded functions to print different data types (int, float, string).

Function overloading can also be used to create multiple functions with the same name but different parameter types.([Learn C++](#))

Example:

```cpp
#include <iostream>

#include <string>


// Overloaded function to print an integer

void print(int i) {

    std::cout << "Integer: " << i << std::endl;

}


// Overloaded function to print a float

void print(float f) {

    std::cout << "Float: " << f << std::endl;

}


// Overloaded function to print a string

void print(const std::string& s) {

    std::cout << "String: " << s << std::endl;

}


int main() {

    print(5);

    print(3.14f);

    print("Hello, World!");

    return 0;

}
```

In this example, the print function is overloaded to print different data types: integer, float, and string.

---

44. Write a function with default arguments to calculate the compound interest.

Default arguments allow functions to be called with fewer arguments than they are defined to accept.

Example:

```cpp
#include <iostream>

#include <cmath> // For pow()


double compoundInterest(double principal, double rate, int time, int n = 1) {

    return principal * std::pow(1 + rate / (100 * n), n * time) - principal;

}


int main() {

    double principal = 1000;

    double rate = 5;

    int time = 2;

    std::cout << "Compound Interest (annually): " << compoundInterest(principal, rate, time) << std::endl;

    std::cout << "Compound Interest (quarterly): " << compoundInterest(principal, rate, time, 4) << std::endl;

    return 0;

}
```

In this example, the compoundInterest function calculates compound interest with a default number of compounding periods per year set to 1 (annually). The default value can be overridden by passing a different value.

---

45. Implement a function with default arguments to print a greeting message (default name is "Guest").

Default arguments allow functions to be called with fewer arguments than they are defined to accept.([Wikitechy](#))

Example:

```cpp
#include <iostream>

#include <string>


void greet(const std::string& name = "Guest") {

    std::cout << "Hello, " << name << "!" << std::endl;

}
```

```cpp
int main() {

    greet();

    greet("Alice");

    return 0;

}
```

In this example, the greet function prints a greeting message. If no name is provided, it defaults to "Guest".

---

46. Create a function that calculates the power of a number with a default exponent of 2.

Default arguments allow functions to be called with fewer arguments than they are defined to accept.

Example:

```cpp
#include <iostream>

#include <cmath> // For pow()


double power(double base, int exponent = 2) {

    return std::pow(base, exponent);

}


int main() {

    std::cout << "3^2 = " << power(3) << std::endl;

    std::cout << "2^3 = " << power(2, 3) << std::endl;

    return 0;

}
```

In this example, the power function calculates the power of a number. If no exponent is provided, it defaults to 2.

---

47. Write a program to demonstrate the concept of recursive functions.

A recursive function is a function that calls itself to solve a problem.

Example:

```cpp
#include <iostream>
```

```
int factorial(int n) {

    if (n <= 1) {

        return 1;

    }

    return n * factorial(n - 1);

}


int main() {

    int number = 5;

    std::cout << "
```

::contentReference{index=71}

Certainly! Let's continue addressing the remaining questions from your assignment:

---

47. Write a program to demonstrate the concept of recursive functions.

A recursive function is one that calls itself to solve a problem. Here's an example demonstrating recursion by calculating the factorial of a number:

```
#include <iostream>


int factorial(int n) {

    if (n <= 1) return 1;

    return n * factorial(n - 1);

}


int main() {

    int number = 5;

    std::cout << "Factorial of " << number << " is " << factorial(number) << std::endl;

    return 0;

}
```

In this example, the factorial function calls itself with a decremented value of n until it reaches the base case (n <= 1).

---

48. Implement a program that uses an array of function pointers.

Function pointers allow you to store addresses of functions and call them dynamically. Here's an example using an array of function pointers to perform basic arithmetic operations:

```cpp
#include <iostream>


int add(int a, int b) { return a + b; }

int subtract(int a, int b) { return a - b; }

int multiply(int a, int b) { return a * b; }

int divide(int a, int b) { return a / b; }


int main() {

    int (*operations[4])(int, int) = {add, subtract, multiply, divide};

    int a = 10, b = 5;

    std::cout << "Addition: " << operations[0](a, b) << std::endl;

    std::cout << "Subtraction: " << operations[1](a, b) << std::endl;

    std::cout << "Multiplication: " << operations[2](a, b) << std::endl;

    std::cout << "Division: " << operations[3](a, b) << std::endl;

    return 0;

}
```

In this example, an array of function pointers operations is used to store addresses of arithmetic functions. The appropriate function is called based on the index in the array.

---

49. Create a program to demonstrate the use of function templates.

Function templates allow you to write generic functions that can operate on different data types. Here's an example using a template to find the maximum of two values:

```cpp
#include <iostream>


template <typename T>

T max(T a, T b) {

    return (a > b) ? a : b;

}
```

```cpp
int main() {

    std::cout << "Max of 3 and 5: " << max(3, 5) << std::endl;

    std::cout << "Max of 3.5 and 2.5: " << max(3.5, 2.5) << std::endl;

    std::cout << "Max of 'a' and 'z': " << max('a', 'z') << std::endl;

    return 0;

}
```

In this example, the max function is defined as a template that can operate on any data type. The compiler generates the appropriate function based on the types of the arguments passed.

---

50. Write a program to illustrate the concept of function pointers and callback functions.

Function pointers can be used to implement callback functions, allowing you to pass functions as arguments to other functions. Here's an example demonstrating this concept:

```cpp
#include <iostream>


void greet() {

    std::cout << "Hello, World!" << std::endl;

}


void executeCallback(void (*callback)()) {

    callback();

}


int main() {

    executeCallback(greet);

    return 0;

}
```

In this example, the executeCallback function takes a function pointer callback as an argument and calls it. The greet function is passed as a callback to executeCallback.