Assignment 2 – Introduction to Programming

* **Overview of C Programming**

1. **Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.**

* C was developed at Bell Labs by Dennis Ritchie and Ken Thompson to improve the UNIX operating system.
* C introduced key features like data types, structures, and greater portability compared to its predecessors.
* C is still used today for building operating systems, embedded devices, and programs where speed is important. Many modern languages like C++ and Python are influenced by C. Even though it's an old language, C is still important because it lets programmers write fast and powerful code.

1. **Research and provide three real-world applications where C programming is extensively used, such as in embedded systems, operating systems, or game development.**

* **Real-World Applications of C Programming** :
* **Embedded Systems** – Used in automotive control units, medical devices, and IoT gadgets due to its efficiency and direct hardware access.
* **Operating Systems** – Major OS like Windows, Linux, and macOS have core components written in C for performance and system control.
* **Game Development** – Game engines like Unreal Engine and physics engines rely on C for high-speed processing and real-time rendering.
* **Setting Up Environment:**

1. **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.**

* Install C Compiler (GCC) & Set Up an IDE
* 1. Install GCC Compiler
* Windows: Install MinGW-w64, add it to the system PATH, and verify with gcc --version.
* Linux/Mac: Install via terminal (sudo apt install build-essential or brew install gcc).
* 2. Set Up an IDE
* Dev-C++: Download from SourceForge, install, and start a new project.
* Code::Blocks: Download with MinGW, install, and select GNU GCC Compiler in settings.
* VS Code: Install, add the C/C++ extension, and configure MinGW for compilation.
* **Basic Structure of a C Program:**

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

 **Headers (#include)** – Import standard libraries (e.g., #include <stdio.h>).

 **Main Function (main())** – Entry point where execution begins.

 **Comments (// or /\* \*/)** – Used for explanations.

 **Data Types** – Define variable types (int, float, char, etc.).

 **Variables** – Store values used in the program.

**Example:**

#include<stdio.h> //Header file

int main() //Main function

{

//Variable declaration

int age = 25;

float height = 5.9;

char grade = ‘A’;

//Output

printf(“Age: %d, Height: %.1f, Grade: %c\n” , age, height,grade);

return 0; //End of program

}

**This Prints:**

* Age: 25, Height: 5.9, Grade:A
* **Operators in C:**

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

* **Arithmetic Operators (Perform mathematical operations)**
* + (Addition), - (Subtraction), \* (Multiplication), / (Division), % (Modulus)
* Example: int sum = 5 + 3; // sum = 8
* **Relational Operators (Compare values, return true (1) or false (0))**
* == (Equal to), != (Not equal to), > (Greater than), < (Less than), >=, <=
* Example: 5 > 3 // true (1)
* **Logical Operators (Used in decision-making)**
* && (AND), || (OR), ! (NOT)
* Example: (5 > 3) && (8 > 6) // true (1)
* **Assignment Operators (Assign values to variables)**
* = (Assign), +=, -=, \*=, /=, %=
* Example: x += 2; // Equivalent to x = x + 2
* **Increment/Decrement Operators (Increase or decrease values)**
* ++ (Increment), -- (Decrement)
* Example: x++ // Increases x by 1
* **Bitwise Operators (Perform operations at the bit level)**
* & (AND), | (OR), ^ (XOR), ~ (NOT), << (Left shift), >> (Right shift)
* Example: 5 & 3 // Output: 1 (Binary: 101 & 011 = 001)
* **Conditional (Ternary) Operator (Shorthand for if-else)**
* condition ? value\_if\_true : value\_if\_false;
* Example: int min = (a < b) ? a : b;
* **Control Flow Statements in C:**

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

**1. if Statement**

* Executes a block of code if a condition is true.
* if (condition) {

// Code to execute if condition is true

}

**Example:**

int num = 10 ;

if(num > 0)

{

printf(“\n\n\tNumber is positive.”);

}

2. if-else Statement:

* Executes one block if the condition is true, otherwise executes another block.
* if (condition) {

// Code if condition is true

} else {

// Code if condition is false

}

Example:

int num = -5;

if (num >= 0) {

printf("Number is positive\n");

} else {

printf("Number is negative\n");

}

3.Nested if-else Statement:

* An if-else inside another if or else.
* if (condition1) {

if (condition2) {

// Code if both conditions are true

} else {

// Code if only condition1 is true

}

} else {

// Code if condition1 is false

}

Example:

* int num = 0;

if (num >= 0) {

if (num == 0) {

printf("Number is zero\n");

} else {

printf("Number is positive\n");

}

} else {

printf("Number is negative\n");

}

4.Switch Statement:

* Used for multiple condition checking (alternative to multiple if-else).

switch (variable) {

case value1:

// Code for case 1

break;

case value2:

// Code for case 2

break;

default:

// Code if no case matches

}

Example:

* int day = 3;

switch (day) {

case 1:

printf("Monday\n");

break;

case 2:

printf("Tuesday\n");

break;

case 3:

printf("Wednesday\n");

break;

default:

printf("Invalid day\n");

}

* **Looping in C:**

**7) Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **while Loop** | **for Loop** | **do-while Loop** |
| **Initialization** | Before the loop | In the loop header | Before the loop |
| **Condition Check** | At the beginning | At the beginning | At the end (executes at least once) |
| **Loop Execution** | Runs only if the condition is true | Runs based on condition | Always executes at least once |
| **Use Case** | When the number of iterations is unknown | When iterations are known | When at least one execution is required |
| **Syntax** | while(condition) {} | for(init; condition; update) {} | do {} while(condition); |

**When to Use Each Loop:**

* **while Loop** → Use when **iterations are unknown** (e.g., reading input until valid).
* **for Loop** → Use when **iterations are known** (e.g., iterating through an array).
* **do-while Loop** → Use when **execution must happen at least once** (e.g., menu-driven programs).
* **Loop Control Statements:**

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

|  |  |  |
| --- | --- | --- |
| **Statement** | **Function** | **Best Use Case** |
| **break** | Exits loop/switch early | Stopping loop when condition met |
| **continue** | skips current iteration | Skipping unwanted values |
| **goto** | Jumps to a label | Handling errors (use sparingly) |

**Examples:**

**1.break Statement:**

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

if (i == 5) break;

printf("%d ", i);

}

**2.continue Statement:**

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

if (i == 5) continue;

printf("%d ", i);

}

**3.goto Statement:**

* if (num < 0) goto error;

error: printf("Invalid number!");

* **Functions in C**

**9)** **What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.**

* Functions is a blocks of code designed to perform a specific task. They help make your code modular, reusable, and easier to maintain. C programs start execution from the main() function, and you can define additional functions as needed.
* **Function Declaration**
* Tells the compiler about the function's name, return type, and parameters.
* int add(int a, int b);
* **Function Definition**
* Contains the actual code of the function

int add(int a, int b) {

return a + b;

}

* **Function Call**
* Invokes the function from main() or another function.

int result = add(3, 4);

* Example:

#include <stdio.h>

int add(int a, int b); // Declaration

int main() {

int sum = add(5, 7); // Call

printf("Sum: %d\n", sum);

return 0;

}

int add(int a, int b) { // Definition

return a + b;

}

* **Arrays in C:**

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

* Array: An **array** is a collection of **same-type elements** stored in **contiguous memory**. You access elements using an **index** (starting from 0).
* **One-Dimensional Array (1D)**
* **Structure**: A simple list.

Example:

int arr[3] = {10, 20, 30};

printf("%d", arr[1]); // Output: 20

* **Multi-Dimensional Array (2D or more)**
* **Structure**: A table (rows × columns).

Example:

int matrix[2][2] = {{1, 2}, {3, 4}};

printf("%d", matrix[1][0]); // Output: 3

* **Pointers in C:**

**11) Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?**

* A **pointer** is a variable that **stores the memory address** of another variable.Think of it as a signpost pointing to the location of a value in memory.
* Declaration and initialization:

int a = 10; // Normal integer variable

int \*p; // Declaration: p is a pointer to int

p = &a; // Initialization: p stores the address of a

* \*p means pointer to an integer.
* &a gives the **address** of variable a.
* \*p (dereferencing) gives the **value** stored at that address.

| **Reason** | **Explanation** |
| --- | --- |
| **Memory Efficiency** | You can access and manipulate memory directly. |
| **Function Arguments (Call by Reference)** | Pass variables by address to modify them in functions. |
| **Dynamic Memory Allocation** | Used with malloc, calloc, free, etc. |
| **Efficient Data Structures** | Crucial for building linked lists, trees, and other structures. |

* **Strings in C:**

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

### ****String Functions (from**** <string.h>****)****

|  |  |  |
| --- | --- | --- |
| **Function** | **Purpose** | **Examples** |
| Strlen(s) | Returns length of string s | strlen("Hello") → 5 |
| strcpy(d, s) | Copies string s to d | strcpy(dest, "World") |
| strcat(d, s) | Appends string s to d | strcat(str1, str2) → Combines both |
| strcmp(s1, s2) | Compares strings | Returns 0 if equal, <0 or >0 otherwise |
| strchr(s, c) | Finds first occurrence of c in s | strchr("hello", 'l') → points to 'l' |

**Use Cases:**

* strlen() → To check string length.
* strcpy() → To duplicate strings.
* strcat() → To build combined messages.
* strcmp() → To compare inputs (like passwords).
* strchr() → To search within strings.
* **Structures in C:**

**13)** **Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.**

* A structure groups different types of variables under one name.

**Declaration:**

struct Student {

char name[50];

int age;

float marks;

};

**Initialization:**

Use the dot operator (.):

printf("%s", s1.name);

**Use Case:**

// Example: Book, Student, Employee, etc.

struct Book {

char title[100];

int pages;

float price;

};

* **File Handling in C:**

**14)** **Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.**

* **Importance of File Handling in C:**
* Allows **permanent storage** of data.
* Useful for **reading/writing** data from/to files.
* Essential for tasks like saving user input, loading configurations, logs, etc.
* **Basic File Operations:**
* All file functions are in the <stdio.h> header.

### ****Opening a File****

### FILE \*fp;

### fp = fopen("data.txt", "w"); // Modes: "r", "w", "a", "r+", etc.

| Mode | Purpose |
| --- | --- |
| "r" | Read (file must exist) |
| "w" | Write (creates or overwrites) |
| "a" | Append |
| "r+" | Read + Write |

1. Writing to a File

fprintf(fp, "Hello, File!\n");

Or character-by-character:

fputc('A', fp);

1. Reading from a File

fclose(fp);

Example Program:

#include <stdio.h>

int main() {

FILE \*fp = fopen("example.txt", "w");

fprintf(fp, "Hello, C File Handling!");

fclose(fp);

fp = fopen("example.txt", "r");

char ch;

while ((ch = fgetc(fp)) != EOF) {

putchar(ch);

}

fclose(fp);

return 0;

}

| **Operation** | **Function** | **Example** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| Open | fopen() | fopen("file.txt", "r") |

|  |  |  |
| --- | --- | --- |
| Read | fgetc(), fscanf() | fgetc(fp) |

|  |  |  |
| --- | --- | --- |
| Write | fputc(), fprintf() | fprintf(fp, ...) |

|  |  |  |
| --- | --- | --- |
| Close | fclose() | fclose(fp) |