**Programming in C - CS3251**

**PART-A**

**UNIT 1 BASICS OF C PROGRAMMING**

**1. (i) Explain in detail about “C” declarations and variables. (Jan 11) (ii) What are constants? Explain the various types of constants in C. (May 15)**

**🔹 C Declarations:**

A declaration in C tells the compiler about the **name and type** of a variable or function.

**Syntax:**

data\_type variable\_name;

**Example:**

int a; // Declaration of an integer variable

float salary; // Declaration of a float variable

You can also initialize variables during declaration:

int marks = 90;

**🔹 Rules for Naming Variables:**

* Must begin with a letter (A–Z, a–z) or underscore (\_)
* Can contain letters, digits (0–9), and underscores
* Cannot be a C keyword (e.g., int, while)
* Case-sensitive (Total ≠ total)

**(ii) Constants in C**

**Constants** are fixed values that **do not change during program execution**. They can be declared using:

* #define directive
* const keyword

**🔹 Types of Constants in C:**

| **Type** | **Example** | **Description** |
| --- | --- | --- |
| **Integer** | 10, -5, 0x2F | Whole numbers (decimal, octal, hex) |
| **Floating-point** | 3.14, -0.5, 2.0e3 | Numbers with decimals or exponential form |
| **Character** | 'A', '9', '\n' | Enclosed in single quotes |
| **String** | "Hello", "123" | Enclosed in double quotes |
| **Enumeration** | enum colors {RED, GREEN}; | Named integer constants |

**Declaring Constants:**

#define PI 3.14 // Preprocessor constant

const int size = 10; // Constant variable

**2. Discuss about the various data types in “C”.**

**Data Types in C**

C provides several built-in data types, classified into:

**🔷 A. Primary Data Types**

| **Data Type** | **Size** | **Description** |
| --- | --- | --- |
| int | 2 or 4 bytes | Integer values |
| float | 4 bytes | Decimal values (single precision) |
| double | 8 bytes | Decimal values (double precision) |
| char | 1 byte | Single characters |
| void | — | Absence of data |

**🔷 B. Derived Data Types**

These are derived from fundamental types:

* Arrays: int arr[5];
* Pointers: int \*ptr;
* Functions: int sum(int a, int b);

**🔷 C. User-Defined Data Types**

* struct: Combines different data types
* union: Memory-efficient version of struct
* enum: Creates symbolic constants
* typedef: Creates new names for existing types

**🔷 D. Qualifiers:**

* **Signed/Unsigned:** unsigned int, signed char
* **Short/Long:** long int, short int

**✅ Summary Table**

| **Concept** | **Description / Example** |
| --- | --- |
| Declaration | int a; → tells compiler about variable name and type |
| Variable | A named memory location to store data |
| Constant | Fixed value using #define or const |
| Data Types | Fundamental (int, char), Derived (Array), User-defined |

**3. (i)Write a C program to check whether the given number is palindrome or not. (May 18) (ii) Write a C program to sum of digits of an integer. (Jan 12, May 14)**

A **palindrome** number is one that remains the same when its digits are reversed.

**🔸 Program:**

#include <stdio.h>

int main() {

int num, reverse = 0, rem, original;

printf("Enter a number: ");

scanf("%d", &num);

original = num;

while (num != 0) {

rem = num % 10;

reverse = reverse \* 10 + rem;

num /= 10;

}

if (original == reverse)

printf("%d is a Palindrome.\n", original);

else

printf("%d is not a Palindrome.\n", original);

return 0;

}

**3. (ii) Sum of Digits of an Integer**

**🔸 Program:**

#include <stdio.h>

int main() {

int num, sum = 0, rem;

printf("Enter a number: ");

scanf("%d", &num);

while (num != 0) {

rem = num % 10;

sum += rem;

num /= 10;

}

printf("Sum of digits = %d\n", sum);

return 0;

}

**4. Write a program to solve the Quadratic equation. (May 15)**

**Program to Solve a Quadratic Equation**

Quadratic equation: **ax² + bx + c = 0**

**🔸 Program:**

#include <stdio.h>

#include <math.h>

int main() {

float a, b, c, d, root1, root2;

printf("Enter coefficients a, b, c: ");

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

d = b \* b - 4 \* a \* c;

if (d > 0) {

root1 = (-b + sqrt(d)) / (2 \* a);

root2 = (-b - sqrt(d)) / (2 \* a);

printf("Roots are real and distinct: %.2f and %.2f\n", root1, root2);

} else if (d == 0) {

root1 = -b / (2 \* a);

printf("Roots are real and equal: %.2f\n", root1);

} else {

float real = -b / (2 \* a);

float imag = sqrt(-d) / (2 \* a);

printf("Roots are complex: %.2f + %.2fi and %.2f - %.2fi\n", real, imag, real, imag);

}

return 0;

}

**5.** **Write a program to find whether a number is prime or not. (May 14)**

**Program to Check for Prime Number**

A prime number is divisible only by 1 and itself.

**🔸 Program:**

#include <stdio.h>

int main() {

int num, i, isPrime = 1;

printf("Enter a number: ");

scanf("%d", &num);

if (num <= 1)

isPrime = 0;

else {

for (i = 2; i <= num / 2; i++) {

if (num % i == 0) {

isPrime = 0;

break;

}

}

}

if (isPrime)

printf("%d is a Prime number.\n", num);

else

printf("%d is not a Prime number.\n", num);

return 0;

}

**6.** **Describe the structure of a C program using “Calculator program” example.**

**Structure of a C Program (Using Calculator Example)**

**🔸 Basic Structure of C Program:**

// Preprocessor Directive

#include <stdio.h>

// Main function

int main() {

// Variable declaration

char op;

float num1, num2, result;

// Input

printf("Enter an operator (+, -, \*, /): ");

scanf(" %c", &op);

printf("Enter two numbers: ");

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

// Switch Case for operation

switch(op) {

case '+':

result = num1 + num2;

printf("Result = %.2f\n", result);

break;

case '-':

result = num1 - num2;

printf("Result = %.2f\n", result);

break;

case '\*':

result = num1 \* num2;

printf("Result = %.2f\n", result);

break;

case '/':

if(num2 != 0)

result = num1 / num2;

else {

printf("Error! Division by zero.\n");

return 1;

}

printf("Result = %.2f\n", result);

break;

default:

printf("Invalid Operator.\n");

}

return 0;

}

**📌 Explanation:**

* **Header files** (#include)
* **main()** function is the entry point
* **Variable declarations** before usage
* **Input/output** using scanf and printf
* **Control structures** (switch, if)
* **Execution ends** with return 0

**UNIT -2 (ARRAYS and STRINGS)**

**1.Explain in detail about One - dimensional Array**

**🔸 Definition:**

A **one-dimensional array** is a linear collection of elements of the same data type stored in contiguous memory locations and accessed using a single index.

**🔸 Syntax:**

datatype arrayName[size];

**🔸 Example:**

int numbers[5] = {10, 20, 30, 40, 50};

**🔸 Accessing Elements:**

printf("%d", numbers[2]); // Output: 30

**🔸 Program Example:**

#include <stdio.h>

int main() {

int marks[5], i;

printf("Enter 5 marks:\n");

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

scanf("%d", &marks[i]);

}

printf("Marks entered:\n");

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

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

}

return 0;

}

**🔸 Uses:**

* Storing a list of values like marks, temperatures, salaries, etc.
* Reducing variable usage and simplifying looping through values.

**2.** **Explain in detail about Two Dimensional Array**

**Two-Dimensional Array in C**

**🔸 Definition:**

A **two-dimensional array** is a table or matrix of elements organized in rows and columns.

**🔸 Syntax:**

datatype arrayName[row][column];

**🔸 Example:**

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

**🔸 Accessing Elements:**

printf("%d", matrix[1][2]); // Output: 6

**🔸 Program Example:**

#include <stdio.h>

int main() {

int matrix[2][2], i, j;

printf("Enter elements of 2x2 matrix:\n");

for(i = 0; i < 2; i++) {

for(j = 0; j < 2; j++) {

scanf("%d", &matrix[i][j]);

}

}

printf("Matrix is:\n");

for(i = 0; i < 2; i++) {

for(j = 0; j < 2; j++) {

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

}

printf("\n");

}

return 0;

}

**🔸 Applications:**

* Matrix operations
* Tabular data representation (e.g., marks of students in subjects)

**3.** **Explain in detail about String Operations**

**🔸 Definition:**

A **string** in C is an array of characters ending with a null character '\0'.

**🔸 Declaring a String:**

char name[10] = "John";

**🔸 Basic String Operations:**

| **Operation** | **Function** | **Example** |
| --- | --- | --- |
| Length | strlen() | strlen("John") → 4 |
| Copy | strcpy() | strcpy(dest, src) |
| Concatenation | strcat() | strcat(str1, str2) |
| Compare | strcmp() | strcmp("a", "b") → -1 |
| Reverse (manual) | No built-in | Use loop to reverse |

**🔸 Program Example (String Operations):**

#include <stdio.h>

#include <string.h>

int main() {

char str1[20], str2[20];

printf("Enter first string: ");

gets(str1);

printf("Enter second string: ");

gets(str2);

printf("Length of str1: %ld\n", strlen(str1));

strcpy(str2, str1);

printf("After copy, str2: %s\n", str2);

strcat(str1, str2);

printf("After concatenation: %s\n", str1);

printf("String comparison: %d\n", strcmp(str1, str2));

return 0;

}

Note: gets() is outdated and unsafe. Use fgets() in real programs.

**✅ Summary Table:**

| **Topic** | **Key Concept** | **Example Keyword** |
| --- | --- | --- |
| One-Dimensional Array | Linear list of elements | arr[i] |
| Two-Dimensional Array | Table/matrix of elements | arr[i][j] |
| String Operations | Functions like strlen, strcpy, etc. | strcat() |

**UNIT 3 FUNCTIONS AND POINTERS**

**1. What do you mean by Call by reference? Explain with an example.**

**🔸 Definition:**

In **Call by Reference**, the **address** of actual parameters is passed to the function. The changes made to the parameters **inside the function reflect** in the actual arguments.

**🔸 Key Points:**

* Function works on the **actual memory location**.
* **Changes are permanent**.
* Implemented using **pointers** in C.

**🔸 Example:**

#include <stdio.h>

void swap(int \*x, int \*y) {

int temp;

temp = \*x;

\*x = \*y;

\*y = temp;

}

int main() {

int a = 10, b = 20;

swap(&a, &b); // passing addresses

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

return 0;

}

**🔸 Output:**

After swap: a = 20, b = 10

**2. What is Call by Value? Explain with an Example**

**🔸 Definition:**

In **Call by Value**, the **copy** of the actual parameter is passed to the function. Changes made to the parameters **inside the function do not affect** the actual values.

**🔸 Key Points:**

* Function works on **copied values**.
* **Original data remains unchanged**.

**🔸 Example:**

#include <stdio.h>

void addTen(int x) {

x = x + 10;

printf("Inside function: x = %d\n", x);

}

int main() {

int a = 5;

addTen(a); // passing value

printf("Outside function: a = %d\n", a);

return 0;

}

**🔸 Output:**

Inside function: x = 15

Outside function: a = 5

**3. Write a C Program to Generate Fibonacci Series using Function**

**🔸 Fibonacci Series:**

0, 1, 1, 2, 3, 5, 8, 13, ...

**🔸 Program:**

#include <stdio.h>

void fibonacci(int n) {

int a = 0, b = 1, next, i;

printf("Fibonacci Series: %d %d ", a, b);

for(i = 2; i < n; i++) {

next = a + b;

printf("%d ", next);

a = b;

b = next;

}

}

int main() {

int n;

printf("Enter number of terms: ");

scanf("%d", &n);

fibonacci(n);

return 0;

}

**🔸 Output:**

Enter number of terms: 6

Fibonacci Series: 0 1 1 2 3 5

**4. Write a C Program to Find Factorial of a Number Using Recursion**

**🔸 Factorial:**

n! = n × (n-1) × (n-2) × ... × 1

**🔸 Program:**

#include <stdio.h>

int factorial(int n) {

if(n == 0)

return 1;

else

return n \* factorial(n - 1);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

printf("Factorial of %d = %d\n", num, factorial(num));

return 0;

}

**🔸 Output:**

Enter a number: 5

Factorial of 5 = 120

**5. What is Pointer? How to pass pointer as an argument in function?**

**✅ 9. What is a Pointer? How to Pass Pointer as an Argument in Function?**

**🔸 What is a Pointer?**

A **pointer** is a variable that stores the **memory address** of another variable.

**🔸 Syntax:**

int \*ptr;

* \*ptr is a pointer to an int.
* It holds the address of an integer variable.

**🔸 Example of Pointer Declaration:**

int a = 10;

int \*p = &a; // p now stores address of a

**🔸 Passing Pointer to a Function**

**➤ Benefits:**

* Helps in modifying the actual value.
* Reduces memory overhead (especially for large data).

**🔸 Example:**

#include <stdio.h>

void updateValue(int \*x) {

\*x = \*x + 10;

}

int main() {

int num = 20;

updateValue(&num); // Passing address

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

return 0;

}

**🔸 Output:**

Updated Value = 30

**🔸 Explanation:**

* &num passes the address of num.
* Inside the function, \*x directly modifies the actual variable.

**6. How Can You Pass an Array to a Function by Value?**

**🔸 Important Note:**

In C, **arrays cannot be passed by value** directly — they are always **passed by reference** by default (i.e., passing the address of the first element).

But, you can **mimic pass-by-value behavior** using **structures** or **manually copying** the array inside the function.

**🔸 Method 1: Normal Pass (Reference-like Behavior)**

#include <stdio.h>

void display(int arr[], int size) {

for(int i = 0; i < size; i++)

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

}

int main() {

int arr[3] = {1, 2, 3};

display(arr, 3);

return 0;

}

**🔸 Output:**

1 2 3

**🔸 Method 2: Mimic Pass by Value using Copy**

#include <stdio.h>

void modifyArray(int arr[], int size) {

int temp[10]; // local copy

for(int i = 0; i < size; i++)

temp[i] = arr[i]; // Copy

temp[0] = 100; // modify copy

printf("Inside function: %d\n", temp[0]);

}

int main() {

int arr[3] = {1, 2, 3};

modifyArray(arr, 3);

printf("Outside function: %d\n", arr[0]); // remains 1

return 0;

}

**🔸 Output:**

Inside function: 100

Outside function: 1

**✅ Summary:**

| **Concept** | **Behavior** |
| --- | --- |
| Pointer | Stores memory address |
| Pointer in function | Allows direct modification of variable |
| Array in C | Passed by reference |
| Pass array by value | Mimic with structure or manual copying |

**UNIT 4 STRUCTURES AND UNION**

**1. What is a structure? Create a structure with data members of various types and declare two structure variables. Write a program to read data into these and print the same. Write short notes on structure Declaration. (Jan 16)**

**🔸 Definition:**

A **structure** in C is a **user-defined data type** that allows combining data items of **different types** under one name.

**🔸 Syntax:**

struct StructureName {

dataType member1;

dataType member2;

...

};

**🔸 Example Structure Declaration and Program:**

#include <stdio.h>

struct Student {

int rollNo;

char name[30];

float marks;

};

int main() {

struct Student s1, s2;

printf("Enter details for Student 1:\n");

printf("Roll No: ");

scanf("%d", &s1.rollNo);

printf("Name: ");

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

printf("Marks: ");

scanf("%f", &s1.marks);

printf("\nEnter details for Student 2:\n");

printf("Roll No: ");

scanf("%d", &s2.rollNo);

printf("Name: ");

scanf("%s", s2.name);

printf("Marks: ");

scanf("%f", &s2.marks);

printf("\nStudent 1 -> Roll: %d, Name: %s, Marks: %.2f\n", s1.rollNo, s1.name, s1.marks);

printf("Student 2 -> Roll: %d, Name: %s, Marks: %.2f\n", s2.rollNo, s2.name, s2.marks);

return 0;

}

**🔸 Short Notes on Structure Declaration:**

* Structure combines different data types.
* Useful in real-time applications like database records.
* Variables of structures are accessed using the dot (.) operator.

**2. What are Self-Referential Structures?**

**🔸 Definition:**

A **self-referential structure** is a structure that **contains a pointer to itself** (same structure type).

**🔸 Used in:**

* Linked Lists
* Trees
* Graphs

**🔸 Example:**

struct Node {

int data;

struct Node\* next; // Self-referential pointer

};

* next points to another node of the same structure.
* This forms the basis for dynamic data structures.

**3. C Program to Implement Singly Linked List using Dynamic Memory Allocation**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

void display(struct Node\* head) {

struct Node\* temp = head;

printf("Linked List: ");

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

struct Node \*head = NULL, \*temp = NULL, \*newNode;

int n, i, value;

printf("Enter number of nodes: ");

scanf("%d", &n);

for (i = 0; i < n; i++) {

newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Memory allocation failed\n");

return 1;

}

printf("Enter data for node %d: ", i + 1);

scanf("%d", &value);

newNode->data = value;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

temp = newNode;

} else {

temp->next = newNode;

temp = newNode;

}

}

display(head);

return 0;

}

**🔸 Output Example:**

Enter number of nodes: 3

Enter data for node 1: 10

Enter data for node 2: 20

Enter data for node 3: 30

Linked List: 10 -> 20 -> 30 -> NULL

**✅ Summary Table**

| **Concept** | **Key Points** |
| --- | --- |
| Structure | Groups variables of different types |
| Self-Referential Structure | Contains a pointer to same structure type |
| Singly Linked List Program | Uses dynamic memory with malloc and pointer manipulation |

UNIT 5 FILE PROCESSING

Here’s a **detailed answer for 16 marks** covering all the five file handling programs in C, along with explanations and code:

**✅ 1. Program to write text to a file from command line arguments**

**📌 Description:**

* Takes filename and a line of text as command line arguments.
* Writes the text to the specified file.

#include <stdio.h>

int main(int argc, char \*argv[]) {

if (argc != 3) {

printf("Usage: %s <filename> <text>\n", argv[0]);

return 1;

}

FILE \*fp = fopen(argv[1], "w");

if (fp == NULL) {

perror("Error opening file");

return 1;

}

fprintf(fp, "%s", argv[2]);

fclose(fp);

printf("Text written to file successfully.\n");

return 0;

}

**✅ 2. Program to copy contents of one file into another**

#include <stdio.h>

int main() {

char ch;

FILE \*source, \*target;

char src[100], dest[100];

printf("Enter source file name: ");

scanf("%s", src);

printf("Enter target file name: ");

scanf("%s", dest);

source = fopen(src, "r");

if (source == NULL) {

perror("Error opening source file");

return 1;

}

target = fopen(dest, "w");

if (target == NULL) {

perror("Error opening target file");

fclose(source);

return 1;

}

while ((ch = fgetc(source)) != EOF)

fputc(ch, target);

printf("File copied successfully.\n");

fclose(source);

fclose(target);

return 0;

}

**✅ 3. Program to append one file to another**

#include <stdio.h>

int main() {

char ch;

FILE \*src, \*dest;

char file1[100], file2[100];

printf("Enter the source file to append: ");

scanf("%s", file1);

printf("Enter the destination file: ");

scanf("%s", file2);

src = fopen(file1, "r");

dest = fopen(file2, "a");

if (src == NULL || dest == NULL) {

perror("Error opening files");

return 1;

}

while ((ch = fgetc(src)) != EOF)

fputc(ch, dest);

printf("File appended successfully.\n");

fclose(src);

fclose(dest);

return 0;

}

**✅ 4. Program to compare two files**

#include <stdio.h>

int main() {

FILE \*file1, \*file2;

char f1[100], f2[100];

char ch1, ch2;

int equal = 1;

printf("Enter first file: ");

scanf("%s", f1);

printf("Enter second file: ");

scanf("%s", f2);

file1 = fopen(f1, "r");

file2 = fopen(f2, "r");

if (file1 == NULL || file2 == NULL) {

perror("Error opening files");

return 1;

}

while (!feof(file1) && !feof(file2)) {

ch1 = fgetc(file1);

ch2 = fgetc(file2);

if (ch1 != ch2) {

equal = 0;

break;

}

}

if (!feof(file1) || !feof(file2))

equal = 0;

fclose(file1);

fclose(file2);

if (equal) {

printf("Files are equal\n");

return 0;

} else {

printf("Files are NOT equal\n");

return 1;

}

}

**5. File Manipulation Functions in C**

**📘 Important Functions:**

| **Function** | **Description** |
| --- | --- |
| fopen() | Opens a file |
| fclose() | Closes an opened file |
| fgetc() | Reads a character from file |
| fputc() | Writes a character to file |
| fgets() | Reads a line of string from file |
| fputs() | Writes a string to a file |
| fread() | Reads block of data from file |
| fwrite() | Writes block of data to file |
| fprintf() | Writes formatted output to file |
| fscanf() | Reads formatted input from file |
| feof() | Checks end-of-file indicator |
| fseek() | Sets file position of stream |
| ftell() | Returns current file pointer position |
| rewind() | Sets file pointer to beginning |

**📌 Example Use:**

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

if (fp != NULL) {

fputs("Hello File!", fp);

fclose(fp);

}