**Programming in C - CS3251**

**PART-C**

**UNIT 1 BASICS OF C PROGRAMMING**

**1. Explain in detail about preprocessor directives with examples.**

**Definition:** Preprocessor directives in C are instructions that are processed by the preprocessor before actual compilation begins. They begin with the symbol #.

**Types of Preprocessor Directives:**

1. **MacroDefinition** Example:
2. #define PI 3.14

This replaces all occurrences of PI with 3.14.

1. **FileInclusion (#include)**  
   Example:
2. #include <stdio.h> // System file
3. #include "myfile.h" // User-defined file
4. **Conditional Compilation (#if, #else, #endif, #ifdef, #ifndef)**  
   Example:
5. #define DEBUG
6. #ifdef DEBUG
7. printf("Debugging is enabled\n");
8. #endif
9. **UndefiningaMacro (#undef)**  
   Example:
10. #undef PI
11. **LineControl (#line)**  
    Example:
12. #line 200 "test.c"

**Purpose:**

* Improve code readability and reusability.
* Allow conditional compilation.
* Separate code into modular files.

**2. Explain the concept of storage classes with suitable example.**

**Storage classes** define the scope (visibility), lifetime, and linkage of variables and/or functions.

1. **auto** (default for local variables)
2. void func() {
3. auto int x = 10;
4. }
5. **register** (stores variable in CPU register if possible)
6. void func() {
7. register int speed = 100;
8. }
9. **static** (retains value between function calls)
10. void counter() {
11. static int count = 0;
12. count++;
13. printf("%d\n", count);
14. }
15. **extern** (refers to a global variable defined in another file) File1.c:
16. int x = 10;

File2.c:

extern int x;

printf("%d", x);

**Working Demonstration:**

#include <stdio.h>

void demo() {

static int x = 0;

x++;

printf("%d\n", x);

}

int main() {

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

demo();

return 0;

}

**Output:**

1

2

3

**3. Write a C program to find the sum of 10 non-negative numbers entered by the user.**

#include <stdio.h>

int main() {

int i, num, sum = 0;

printf("Enter 10 non-negative numbers:\n");

for(i = 0; i < 10; i++) {

scanf("%d", &num);

if(num < 0) {

printf("Invalid input. Please enter a non-negative number.\n");

i--; // retry the same iteration

continue;

}

sum += num;

}

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

return 0;

}

**Explanation:**

* The loop runs 10 times.
* If a negative number is entered, the user is prompted again.
* Valid numbers are added to the sum.

UNIT -2 (ARRAYS and STRINGS)

**1. Explain Selection Sort in C**

**Definition:** Selection Sort is a simple comparison-based sorting algorithm. It divides the array into two parts: sorted and unsorted. It repeatedly selects the smallest (or largest) element from the unsorted part and moves it to the sorted part.

**Algorithm Steps:**

1. Start from index 0.
2. Find the smallest element in the unsorted array.
3. Swap it with the element at the beginning.
4. Move the boundary of the sorted array one step forward.
5. Repeat until the array is completely sorted.

**C Program:**

#include <stdio.h>

void selectionSort(int arr[], int n) {

int i, j, min\_idx, temp;

for (i = 0; i < n-1; i++) {

min\_idx = i;

for (j = i+1; j < n; j++) {

if (arr[j] < arr[min\_idx])

min\_idx = j;

}

// Swap

temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

}

}

void printArray(int arr[], int n) {

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

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

printf("\n");

}

int main() {

int arr[] = {64, 25, 12, 22, 11};

int n = sizeof(arr)/sizeof(arr[0]);

printf("Original array:\n");

printArray(arr, n);

selectionSort(arr, n);

printf("Sorted array:\n");

printArray(arr, n);

return 0;

}

**Output:**

Original array:

64 25 12 22 11

Sorted array:

11 12 22 25 64

**2. Explain Linear Search in C**

**Definition:** Linear Search is the simplest search algorithm. It checks each element of the array one by one until the desired element is found or the end is reached.

**Algorithm Steps:**

1. Start from index 0.
2. Compare each element with the key.
3. If found, return the index.
4. If not found by end of array, return -1.

**C Program:**

#include <stdio.h>

int linearSearch(int arr[], int n, int key) {

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

if (arr[i] == key)

return i;

}

return -1;

}

int main() {

int arr[] = {5, 8, 3, 6, 2};

int n = sizeof(arr)/sizeof(arr[0]);

int key = 6;

int result = linearSearch(arr, n, key);

if (result != -1)

printf("Element found at index %d\n", result);

else

printf("Element not found\n");

return 0;

}

**Output:**

Element found at index 3

**3. Explain Binary Search in C**

**Definition:** Binary Search is an efficient search algorithm that works only on sorted arrays. It repeatedly divides the search interval in half and compares the middle element with the key.

**Algorithm Steps:**

1. Set low = 0 and high = n-1.
2. Find mid: mid = (low + high)/2.
3. If arr[mid] == key, return mid.
4. If arr[mid] > key, search in the left half.
5. If arr[mid] < key, search in the right half.
6. Repeat until low > high.

**C Program:**

#include <stdio.h>

int binarySearch(int arr[], int n, int key) {

int low = 0, high = n - 1, mid;

while (low <= high) {

mid = (low + high) / 2;

if (arr[mid] == key)

return mid;

else if (arr[mid] < key)

low = mid + 1;

else

high = mid - 1;

}

return -1;

}

int main() {

int arr[] = {2, 4, 6, 8, 10, 12};

int n = sizeof(arr)/sizeof(arr[0]);

int key = 8;

int result = binarySearch(arr, n, key);

if (result != -1)

printf("Element found at index %d\n", result);

else

printf("Element not found\n");

return 0;

}

**Output:**

Element found at index 3

**UNIT 3 FUNCTIONS AND POINTERS**

**1. (i) Explain the purpose of a function prototype. And specify the difference between the**

**user defined function and built-in function (May 18)**

**(ii) Write the C program to find the value of sin(x) using the series up to the given**

**accuracy (without using user defined function) also print sin(x) using library function.**

**(May 18)**

**1. (i) Purpose of a Function Prototype & Differences Between User-Defined and Built-in Functions**

**Function Prototype:**

* A function prototype is a declaration of a function that tells the compiler about the function name, return type, and parameters before its actual definition.
* It enables the compiler to perform **type checking** of function calls.

**Syntax:**

return\_type function\_name(parameter\_list);

**Example:**

float add(float, float);

**Benefits of Function Prototypes:**

* Helps avoid type mismatch errors.
* Allows calling a function before it is defined.
* Useful in large programs with multiple files.

**User-Defined Function vs Built-in Function:**

| **Feature** | **User-Defined Function** | **Built-in Function** |
| --- | --- | --- |
| Defined by | Programmer | C Standard Library |
| Requires prototype | Yes | Already defined |
| Example | float area(float r) | printf(), sin(), sqrt() |
| Flexibility | Fully customizable | Limited to standard operations |

**1. (ii) C Program to Calculate sin(x) Using Series and Built-in Function**

**Formula:**

sin⁡(x)=x−x33!+x55!−x77!+…\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots

**C Program:**

#include <stdio.h>

#include <math.h>

int main() {

float x, term, sum;

int i, n;

printf("Enter the value of x (in radians): ");

scanf("%f", &x);

term = x;

sum = x;

n = 1;

while (fabs(term) >= 0.0001) {

term = -term \* x \* x / ((2 \* n) \* (2 \* n + 1));

sum += term;

n++;

}

printf("sin(%f) using series = %f\n", x, sum);

printf("sin(%f) using built-in function = %f\n", x, sin(x));

return 0;

}

**Output Example:**

Enter the value of x (in radians): 0.5

sin(0.5) using series = 0.479426

sin(0.5) using built-in function = 0.479426

**2. (i) Difference Between Pass by Value and Pass by Reference + Swap Program**

| **Feature** | **Pass by Value** | **Pass by Reference** |
| --- | --- | --- |
| Copies | Copy of variable passed | Address of variable passed |
| Modifies original | No | Yes |
| Syntax | Default in C | Done using pointers |

**Swapping Using Pass by Reference in C:**

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 10, y = 20;

printf("Before swapping: x = %d, y = %d\n", x, y);

swap(&x, &y);

printf("After swapping: x = %d, y = %d\n", x, y);

return 0;

}

**2. (ii) Recursion and Computing sin(x) Using Recursive Function**

**What is Recursion?**

Recursion is a programming technique where a function calls itself to solve a problem.

**Recursive Formula for sin(x):**

sin⁡(x,n)=x−x33!+x55!−… up to n terms\sin(x, n) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \text{ up to n terms}

**C Code Using Recursion:**

#include <stdio.h>

#include <math.h>

float power(float x, int y) {

if (y == 0)

return 1;

return x \* power(x, y - 1);

}

int fact(int n) {

if (n == 0)

return 1;

return n \* fact(n - 1);

}

float sinx(float x, int n) {

if (n == 0)

return 0;

int sign = (n % 2 == 0) ? -1 : 1;

int exp = 2 \* n - 1;

return sinx(x, n - 1) + sign \* power(x, exp) / fact(exp);

}

int main() {

float x;

int terms;

printf("Enter x in radians and number of terms: ");

scanf("%f %d", &x, &terms);

printf("sin(%f) using recursion = %f\n", x, sinx(x, terms));

printf("sin(%f) using built-in = %f\n", x, sin(x));

return 0;

}

**Sample Output:**

Enter x in radians and number of terms: 0.5 5

sin(0.5) using recursion = 0.479425

sin(0.5) using built-in = 0.479426

**UNIT 4 STRUCTURES AND UNION**

**1. (i) Write short notes on nested structure / Explain the concept of structure within structure with suitable program. (ii) Define and declare a structure to store date, which including day, month and year. (Jan 14)**

**Nested Structure in C (Structure within Structure)**

**Definition:**

A **nested structure** in C is a structure that contains another structure as its member. This helps in logically grouping related data together.

**Syntax:**

struct Inner {

int a;

};

struct Outer {

struct Inner in;

int b;

};

**Example Program:**

#include <stdio.h>

struct Date {

int day;

int month;

int year;

};

struct Student {

char name[50];

int rollNo;

struct Date dob; // Nested structure

};

int main() {

struct Student s;

printf("Enter name: ");

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

printf("Enter roll number: ");

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

printf("Enter date of birth (dd mm yyyy): ");

scanf("%d %d %d", &s.dob.day, &s.dob.month, &s.dob.year);

printf("\nStudent Details:\n");

printf("Name: %s\nRoll No: %d\nDOB: %02d-%02d-%04d\n",

s.name, s.rollNo, s.dob.day, s.dob.month, s.dob.year);

return 0;

}

**1. (ii) Define and Declare a Structure to Store Date (Jan 2014)**

**Structure Declaration:**

struct Date {

int day;

int month;

int year;

};

**Usage:**

int main() {

struct Date today;

today.day = 14;

today.month = 4;

today.year = 2025;

printf("Today's Date: %02d-%02d-%04d\n", today.day, today.month, today.year);

return 0;

}

**2. Structure for Book Details (May 2015)**

**Definition of Structure:**

struct Book {

char name[100];

char author[100];

float price;

};

**Complete Program to Read and Display Book Details:**

#include <stdio.h>

#include <string.h>

struct Book {

char name[100];

char author[100];

float price;

};

int main() {

struct Book books[200];

int i, n;

float total = 0;

printf("Enter number of books (max 200): ");

scanf("%d", &n);

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

printf("\nEnter details of book %d\n", i + 1);

printf("Book Name: ");

scanf(" %[^\n]", books[i].name); // Read string with spaces

printf("Author Name: ");

scanf(" %[^\n]", books[i].author);

printf("Price: ");

scanf("%f", &books[i].price);

total += books[i].price;

}

printf("\nTotal cost of all books = Rs. %.2f\n", total);

printf("\nBooks priced above Rs.500:\n");

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

if (books[i].price > 500) {

printf("Book Name: %s\nAuthor: %s\nPrice: %.2f\n\n",

books[i].name, books[i].author, books[i].price);

}

}

return 0;

}

**Sample Output:**

Enter number of books (max 200): 2

Enter details of book 1

Book Name: C Programming

Author Name: Dennis Ritchie

Price: 450

Enter details of book 2

Book Name: Data Structures

Author Name: Mark Allen

Price: 600

Total cost of all books = Rs. 1050.00

Books priced above Rs.500:

Book Name: Data Structures

Author: Mark Allen

Price: 600.00

**UNIT 5 FILE PROCESSING**

**1. Write the case study of “How sequential Access file is differing from Random Access file”. (May 18)**

**Case Study: Sequential Access File vs Random Access File (May 18)**

**Sequential Access File:**

* **Definition:** Data is accessed **in the order** in which it is stored.
* **Usage:** Best when you need to read or process **all records from start to end**.
* **Example Functions:** fopen(), fgets(), fscanf(), fread(), etc.

**Random Access File:**

* **Definition:** Allows access to any part of the file **directly without reading the previous data**.
* **Usage:** Best when you need to **search, update, or modify** specific records.
* **Example Functions:** fseek(), ftell(), rewind(), etc.

**Case Study Example:**

Suppose you have an employee record file.

**Sequential Access Use Case:**

You want to print a **monthly salary report** of all employees.

**Random Access Use Case:**

You want to **update the salary** of only one employee without reading all records.

| **Feature** | **Sequential File** | **Random Access File** |
| --- | --- | --- |
| Access Method | One-by-one (start to end) | Directly any record |
| Speed | Slower for large data | Faster access |
| File pointer manipulation | Not required | fseek(), ftell() required |
| Use Case | Reports, logs | Database, employee management |

**2. C Program using fwrite() and fread() (May 18)**

#include <stdio.h>

#include <stdlib.h>

struct Student {

int roll;

char name[50];

float marks;

};

int main() {

struct Student s[2], s2[2];

FILE \*fp;

int i;

// Write to file

fp = fopen("student.dat", "wb");

if (fp == NULL) {

printf("Error opening file.\n");

return 1;

}

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

printf("Enter roll, name and marks of student %d:\n", i + 1);

scanf("%d %s %f", &s[i].roll, s[i].name, &s[i].marks);

}

fwrite(s, sizeof(struct Student), 2, fp);

fclose(fp);

// Read from file

fp = fopen("student.dat", "rb");

fread(s2, sizeof(struct Student), 2, fp);

printf("\nStudent Records from File:\n");

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

printf("Roll: %d, Name: %s, Marks: %.2f\n", s2[i].roll, s2[i].name, s2[i].marks);

}

fclose(fp);

return 0;

}

**3. File Operations in C with Examples (May 19)**

**1. Opening a File**

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

**2. Writing to a File**

fprintf(fp, "Hello World");

fputs("Welcome", fp);

fwrite(&data, sizeof(data), 1, fp);

**3. Reading from a File**

fscanf(fp, "%s", str);

fgets(str, 100, fp);

fread(&data, sizeof(data), 1, fp);

**4. Closing a File**

fclose(fp);

**5. File Pointer Manipulation**

* fseek(fp, offset, origin)
* ftell(fp) – Returns current position
* rewind(fp) – Goes to beginning

**6. Error Handling**

if (fp == NULL) {

printf("File cannot be opened.\n");

}

**4. Random Access in Files using fseek(), ftell(), rewind() (May 19)**

**Random Access Functions:**

| **Function** | **Description** |
| --- | --- |
| fseek() | Moves the file pointer |
| ftell() | Returns current file position |
| rewind() | Moves pointer to beginning of file |

**Syntax:**

fseek(FILE \*fp, long int offset, int origin);

* offset: Number of bytes to move
* origin: SEEK\_SET, SEEK\_CUR, SEEK\_END

**Example Program – Modify Record in File**

#include <stdio.h>

#include <stdlib.h>

struct Employee {

int id;

float salary;

};

int main() {

struct Employee e;

FILE \*fp = fopen("emp.dat", "rb+");

if (!fp) {

printf("Cannot open file\n");

return 1;

}

// Move to 2nd record (index 1)

fseek(fp, sizeof(struct Employee) \* 1, SEEK\_SET);

// Modify record

e.id = 102;

e.salary = 45000;

fwrite(&e, sizeof(struct Employee), 1, fp);

rewind(fp); // Optional: move back to start for further ops

fclose(fp);

return 0;

}