

# Programming In C (4331105) - Winter 2024 Solution

Milav Dabgar

May 21, 2024

## Question 1(a) [3 marks]

List any six keywords of C language.

### Solution

Table 1. Keywords Categories

Category	Keywords
Data Types	int, float, char
Control Flow	if, for, return

### Mnemonic

"I Find Clever Reasons For Results"

## Question 1(b) [4 marks]

Define Operator. Summarize types of operators based on operands.

### Solution

**Operator:** Symbol that performs operations on operands to produce a result.

Table 2. Types of Operators

Type	Description	Examples
Unary	Single operand	++, --, !
Binary	Two operands	+, -, *, /, %
Ternary	Three operands	?:

### Mnemonic

"U-B-T: Use Binary Then Ternary"

## Question 1(c) [7 marks]

Define flowchart. Draw flowchart symbols. Draw flowchart to find minimum of two integer numbers N1 & N2.

### Solution

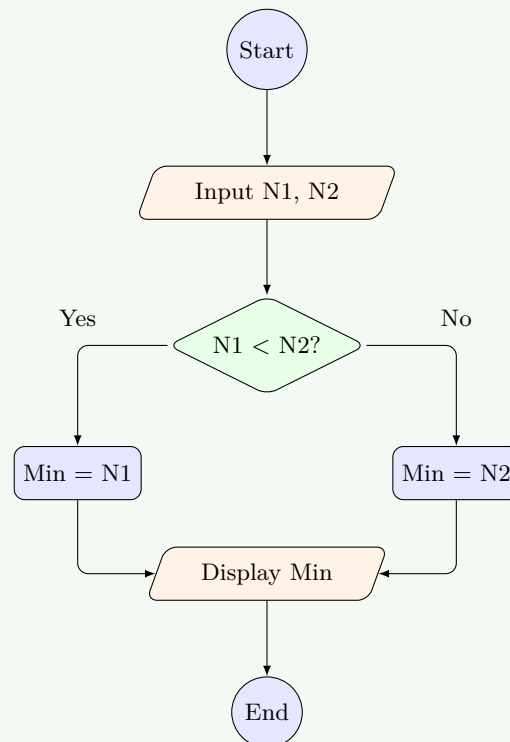
**Flowchart:** Graphical representation of algorithm using standardized symbols to show the sequence of operations.

**Common Flowchart Symbols:**

**Table 3.** Flowchart Symbols

Symbol	Meaning
Oval	Start/Stop
Parallelogram	Input/Output
Rectangle	Process
Diamond	Decision
Arrow	Flow direction

**Flowchart to find minimum of N1 & N2:**



**Figure 1.** Flowchart for Minimum of Two Numbers

### Mnemonic

“SPADE: Start-Process-Arrow-Decision-End”

OR

### Question 1(c) [7 marks]

Define algorithm. Write an algorithm to calculate area and circumference of circle.

### Solution

**Algorithm:** Step-by-step procedure to solve a problem using finite number of well-defined instructions.

**Algorithm to calculate area and circumference of circle:**

1. Start
2. Input radius  $r$
3. Set  $PI = 3.14159$
4. Calculate  $area = PI \times r \times r$
5. Calculate  $circumference = 2 \times PI \times r$
6. Display area and circumference
7. Stop

Table 4. Table of formulas used

Measurement	Formula
Area	$\pi \times r^2$
Circumference	$2 \times \pi \times r$

**Mnemonic**

“RICARD: Radius Input, Calculate Area, Reveal Dimensions”

**Question 2(a) [3 marks]**

Differentiate `printf()` and `scanf()`.

**Solution**Table 5. Difference between `printf()` and `scanf()`

Feature	<code>printf()</code>	<code>scanf()</code>
Purpose	Outputs data to screen	Inputs data from keyboard
Direction	Output function	Input function
Format specifier	Required	Required
Parameter	Actual values	Address of variables (&)

**Mnemonic**

“OIAD: Output-Input, Actual-Destination”

**Question 2(b) [4 marks]**

Develop a C program to print sum & average of 1 to  $n$ .

**Solution**

Listing 1. Sum and Average Program

```

1  #include <stdio.h>
2
3  int main() {
4      int n, i, sum = 0;
5      float avg;
6
7      printf("Enter n: ");
8      scanf("%d", &n);
9  }
```

```

10     for(i = 1; i <= n; i++) {
11         sum += i;
12     }
13
14     avg = (float)sum / n;
15
16     printf("Sum = %d\n", sum);
17     printf("Average = %.2f\n", avg);
18
19     return 0;
20 }

```

**Key Points:**

- **Initialization:** sum = 0
- **Iteration:** for loop from 1 to n
- **Type Casting:** (float) for correct average

**Mnemonic**

“SIAP: Sum Initialize, Add in loop, Print results”

**Question 2(c) [7 marks]**

Explain Arithmetic operator and Relational operator with example.

**Solution****1. Arithmetic Operators:****Table 6.** Arithmetic Operators

Operator	Operation	Example	Result
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	5 / 2	2 (integer)
%	Modulo (Remainder)	5 % 2	1

**2. Relational Operators:****Table 7.** Relational Operators

Operator	Meaning	Example	Result
<	Less than	5 < 3	0 (false)
>	Greater than	5 > 3	1 (true)
<=	Less than or equal	5 <= 5	1 (true)
>=	Greater than or equal	3 >= 5	0 (false)
==	Equal to	5 == 5	1 (true)
!=	Not equal to	5 != 3	1 (true)

**Code Example:**

```

1  int a = 5, b = 3;
2  printf("a + b = %d\n", a + b);    // Output: 8
3  printf("a > b is %d\n", a > b);    // Output: 1 (true)

```

**Mnemonic**

“ASMDR for Arithmetic, LEGENE for Relational”

OR

**Question 2(a) [3 marks]**

What is the difference between `get(S)` and `scanf("%s",S)`

**Solution**

**Table 8.** Difference between `gets(S)` and `scanf()`

Feature	<code>gets(S)</code>	<code>scanf("%s",S)</code>
<b>Whitespace handling</b>	Reads space	Stops at whitespace
<b>Buffer overflow</b>	No boundary check	Safer with width limit
<b>Return type</b>	<code>char*</code>	Number of items read
<b>Usage safety</b>	Deprecated, unsafe	Safer with format control

**Mnemonic**

“WBRU: Whitespace-Boundary-Return-Usage”

OR

**Question 2(b) [4 marks]**

Develop a C program to swap (exchange) value of two numbers.

**Solution**

**Listing 2.** Swap Two Numbers

```

1  #include <stdio.h>
2
3  int main() {
4      int a, b, temp;
5
6      printf("Enter two numbers: ");
7      scanf("%d %d", &a, &b);
8
9      printf("Before swap: a = %d, b = %d\n", a, b);
10
11     temp = a;
12     a = b;
13     b = temp;
14
15     printf("After swap: a = %d, b = %d\n", a, b);
16
17     return 0;
18 }
```

**Diagram:**

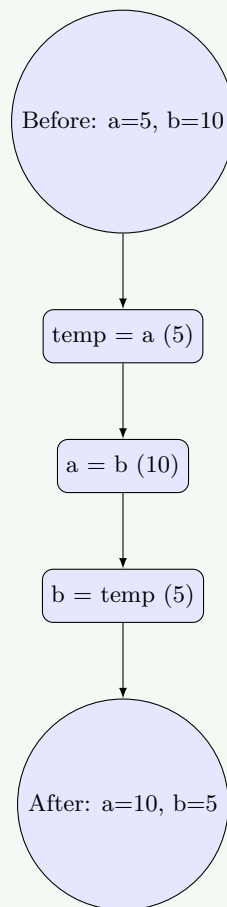


Figure 2. Swapping Logic using Temp Variable

**Mnemonic**

“TAB: Temp-Assign-Backfill”

**OR****Question 2(c) [7 marks]****Explain Boolean operator and Logical operator with example.****Solution****1. Boolean Operators:****Table 9.** Boolean (Bitwise) Operators

Operator	Operation	Example	Result
&	Bitwise AND	5 & 3	1
	Bitwise OR	5   3	7
^	Bitwise XOR	5 ^ 3	6
~	Bitwise NOT	~5	-6
«	Left Shift	5 « 1	10
»	Right Shift	5 » 1	2

## 2. Logical Operators:

Table 10. Logical Operators

Operator	Meaning	Example	Result
&&	Logical AND	(5>3) && (2<4)	1 (true)
	Logical OR	(5<3)    (2<4)	1 (true)
!	Logical NOT	!(5>3)	0 (false)

### Example:

```

1 int a = 5, b = 3;
2 printf("a & b = %d\n", a & b);           // Output: 1 (bitwise AND)
3 printf("a > b && b < 10 is %d\n", a > b && b < 10); // Output: 1 (true)

```

### Bit Representation (5 & 3):

```

1 5 = 101
2 3 = 011
3 & = 001 (1 in decimal)

```

### Mnemonic

“BOXNRL for Boolean, AON for Logical”

## Question 3(a) [3 marks]

Compare entry controlled and exit controlled loop with example.

### Solution

Table 11. Entry vs Exit Controlled Loop

Feature	Entry Controlled	Exit Controlled
Condition check	Before execution	After execution
Minimum iterations	Zero	One
Example	while, for	do-while
Usage	When pre-check needed	When at least one execution needed

### Mnemonic

“BCME: Before-Check-Multiple-Examples”

## Question 3(b) [4 marks]

Develop a C program to display addition and subtraction of two numbers using switch case.

### Solution

Listing 3. Switch Case Calculator

```

1  #include <stdio.h>
2
3  int main() {
4      int a, b, choice, result;
5
6      printf("Enter two numbers: ");
7      scanf("%d %d", &a, &b);
8
9      printf("1. Addition\n2. Subtraction\n");
10     printf("Enter choice (1/2): ");
11     scanf("%d", &choice);
12
13     switch(choice) {
14         case 1:
15             result = a + b;
16             printf("Addition: %d\n", result);
17             break;
18         case 2:
19             result = a - b;
20             printf("Subtraction: %d\n", result);
21             break;
22         default:
23             printf("Invalid choice\n");
24     }
25
26     return 0;
27 }

```

Flowchart:

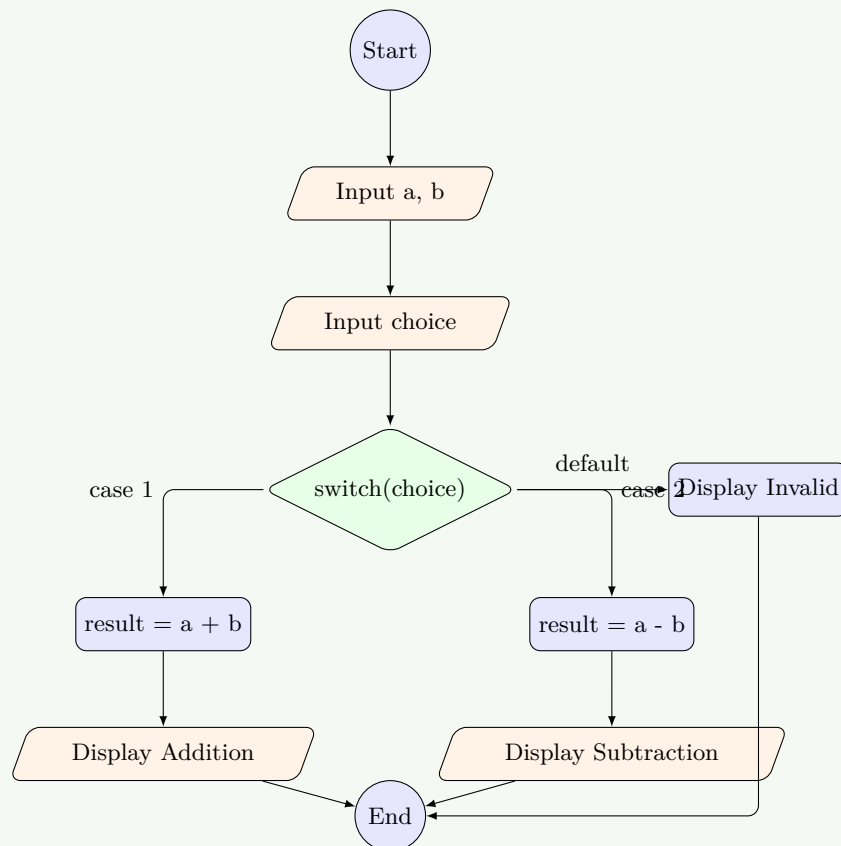


Figure 3. Flowchart for Switch Case Operation



**Mnemonic**

“CIRCA: Choice-Input-Result-Calculate-Action”

**Question 3(c) [7 marks]**

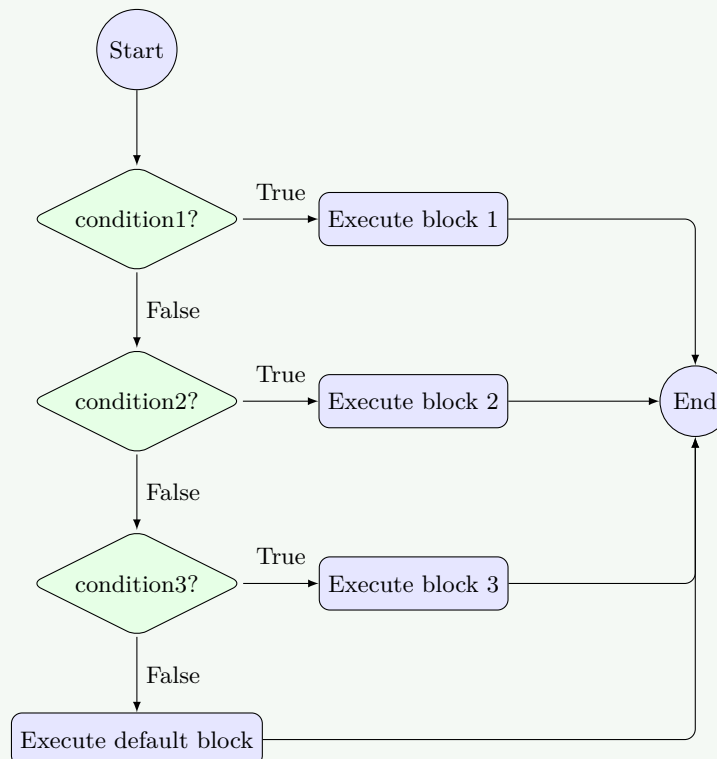
Explain multiple if-else statement with syntax, flowchart and an example.

**Solution****Syntax of multiple if-else:**

```

1  if (condition1) {
2      // code block 1
3  }
4  else if (condition2) {
5      // code block 2
6  }
7  else if (condition3) {
8      // code block 3
9  }
10 else {
11     // default code block
12 }

```

**Flowchart:**

**Figure 4.** Multiple If-Else Flowchart

**Example:**

```

1  #include <stdio.h>
2
3  int main() {
4      int marks;

```

```

5
6     printf("Enter marks: ");
7     scanf("%d", &marks);
8
9     if (marks >= 80) {
10        printf("Grade: A\n");
11    }
12    else if (marks >= 70) {
13        printf("Grade: B\n");
14    }
15    else if (marks >= 60) {
16        printf("Grade: C\n");
17    }
18    else {
19        printf("Grade: F\n");
20    }
21
22    return 0;
23 }

```

**Mnemonic**

“TEST: Try Each Statement Then default”

OR

**Question 3(a) [3 marks]**

State the use of break and continue keyword.

**Solution**

Table 12. Break vs Continue

Keyword	Purpose	Effect	Common Use
<b>break</b>	Terminates loop/switch	Exits the current loop/switch	To exit when condition met
<b>continue</b>	Skips iteration	Jumps to next iteration	To skip specific values

**Example Code:**

```

1 // break example
2 for(i=1; i<=10; i++) {
3     if(i == 5) break; // exits loop at i=5
4     printf("%d ", i); // prints 1 2 3 4
5 }
6
7 // continue example
8 for(i=1; i<=5; i++) {
9     if(i == 3) continue; // skips i=3
10    printf("%d ", i); // prints 1 2 4 5
11 }

```

**Mnemonic**

“EXIT-SKIP: EXit IT or SKIP iteration”

OR

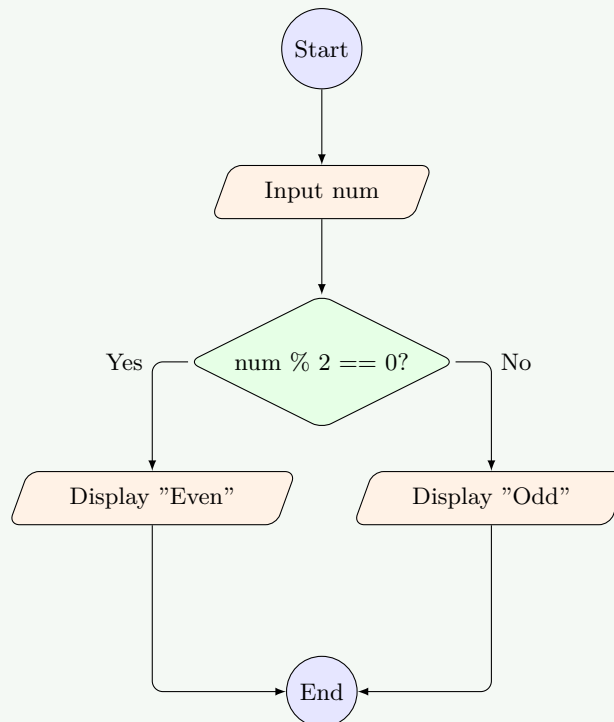
**Question 3(b) [4 marks]**

Develop a C program to check whether the given number is even or odd.

**Solution****Listing 4. Even Odd Check**

```
1 #include <stdio.h>
2
3 int main() {
4     int num;
5
6     printf("Enter a number: ");
7     scanf("%d", &num);
8
9     if (num % 2 == 0) {
10        printf("%d is even.\n", num);
11    }
12    else {
13        printf("%d is odd.\n", num);
14    }
15
16    return 0;
17 }
```

Diagram:

**Figure 5. Flowchart for Even/Odd Check****Key Points:**

- **Check:** Using modulo (%) operator
- **Decision:** Based on remainder with 2
- **Output:** Even for remainder 0, Odd otherwise

**Mnemonic**

“MODE: MODulo Equals zero for even”

OR

### Question 3(c) [7 marks]

Explain switch-case statement with syntax, flowchart and an example.

**Solution****Syntax of switch-case:**

```

1  switch (expression) {
2      case constant1:
3          // code block 1
4          break;
5      case constant2:
6          // code block 2
7          break;
8      ...
9      default:
10         // default code block
11 }

```

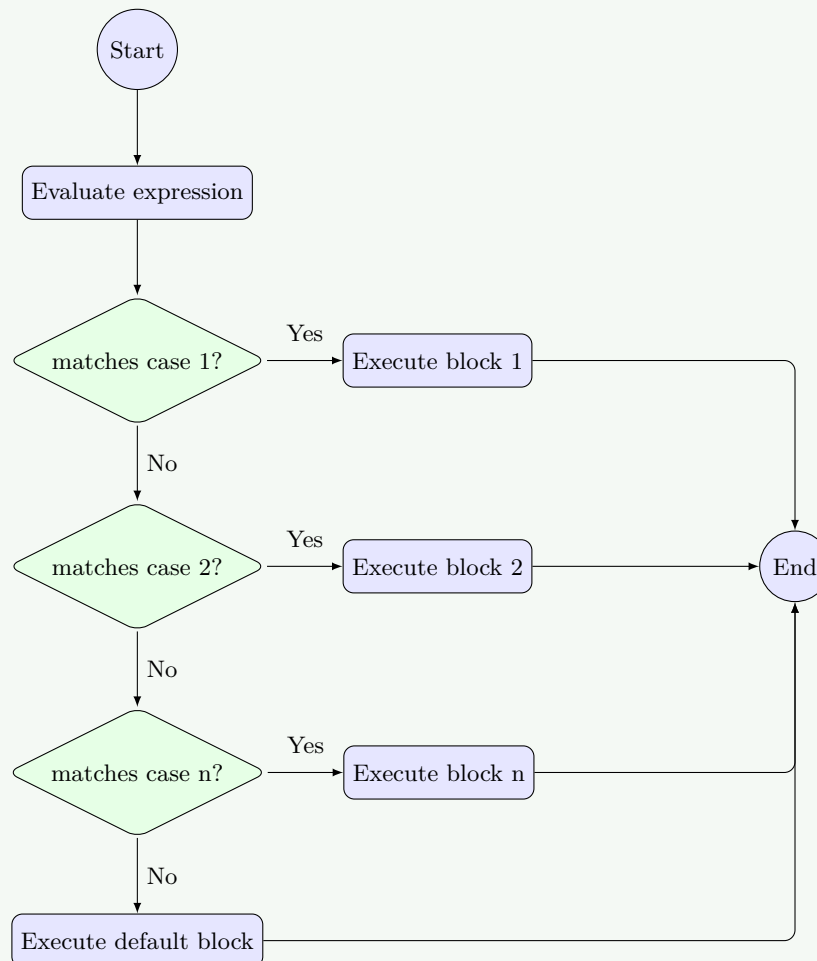
**Flowchart:**

Figure 6. Switch Case Flowchart

**Example:**

```

1  #include <stdio.h>
2
3  int main() {
4      char grade;
5
6      printf("Enter grade (A-D): ");
7      scanf(" %c", &grade);
8
9      switch (grade) {
10         case 'A':
11             printf("Excellent!\n");
12             break;
13         case 'B':
14             printf("Good job!\n");
15             break;
16         case 'C':
17             printf("Satisfactory\n");
18             break;
19         case 'D':
20             printf("Needs improvement\n");
21             break;
22         default:
23             printf("Invalid grade\n");
24     }
25
26     return 0;
27 }

```

**Mnemonic**

“CEBID: Compare-Execute-Break-If-Done”

**Question 4(a) [3 marks]**

Define string. List out different operations that can be performed on string.

**Solution**

**String:** Array of characters terminated by null character '\0'.

**Table 13.** String Operations

Operation	Description	Function
Input/Output	Read/write strings	gets(), puts()
Copy	Copy one string to another	strcpy()
Concatenation	Join two strings	strcat()
Comparison	Compare two strings	strcmp()
Length	Find string length	strlen()
Search	Find substring	strstr()

**Mnemonic**

“ICCLS: Input-Copy-Concatenate-Length-Search”

**Question 4(b) [4 marks]**

Develop a C program to convert uppercase alphabet to lowercase alphabet.

**Solution****Listing 5.** Uppercase to Lowercase

```

1  #include <stdio.h>
2
3  int main() {
4      char ch;
5
6      printf("Enter an uppercase letter: ");
7      scanf(" %c", &ch);
8
9      if (ch >= 'A' && ch <= 'Z') {
10         char lowercase = ch + 32; // ASCII difference is 32
11         printf("Lowercase: %c\n", lowercase);
12     }
13     else {
14         printf("Not an uppercase letter\n");
15     }
16
17     return 0;
18 }

```

ASCII Table Excerpt:

**Table 14.** ASCII Values

Character	ASCII Value
A	65
a	97
Z	90
z	122
Difference	32

**Mnemonic**

“COOL: Character Offset Of Lowercase”

**Question 4(c) [7 marks]**

Draw flowchart of for loop and explain with example.

**Solution**

**For Loop Syntax:**

```

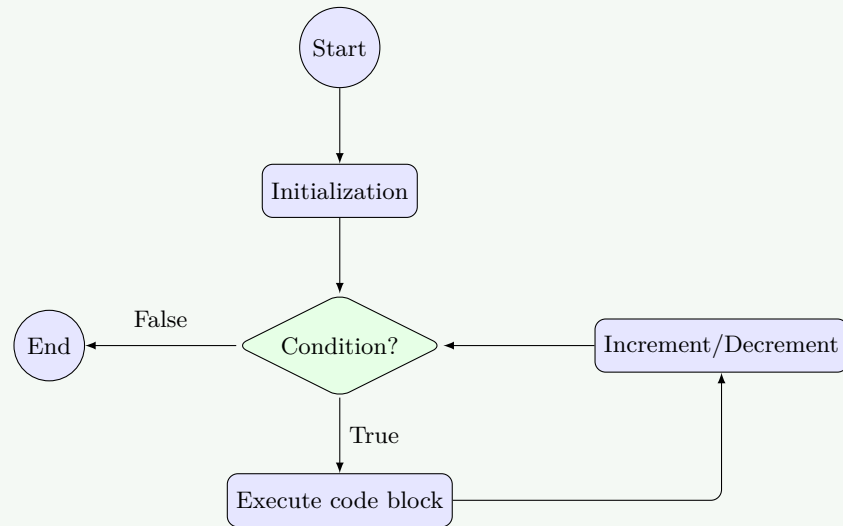
1  for (initialization; condition; increment/decrement) {

```

```

2 // code block
3 }

```

**Flowchart:****Figure 7.** For Loop Flowchart**For Loop Components:**

1. **Initialization:** Executed once at beginning
2. **Condition:** Checked before each iteration
3. **Increment/Decrement:** Executed after each iteration
4. **Body:** Executed if condition is true

**Example:**

```

1 #include <stdio.h>
2
3 int main() {
4     int i;
5
6     for (i = 1; i <= 5; i++) {
7         printf("%d ", i);
8     }
9     // Output: 1 2 3 4 5
10
11     return 0;
12 }

```

**Execution Flow:**

1. Initialize  $i = 1$
2. Check condition  $(1 \leq 5)$  - True
3. Execute body - Print 1
4. Increment  $i$  to 2
5. Check condition  $(2 \leq 5)$  - True
6. And so on until  $i$  becomes 6

**Mnemonic**

“ICE-T: Initialize, Check, Execute, Then increment”

OR

### Question 4(a) [3 marks]

Define array. List out different operations that can be performed on array.

#### Solution

**Array:** Collection of similar data types stored in contiguous memory locations.

**Table 15.** Array Operations

Operation	Description	Example
<b>Declaration</b>	Create array	<code>int arr[5];</code>
<b>Initialization</b>	Assign values	<code>arr[0] = 10;</code>
<b>Traversal</b>	Access all elements	<code>for loop</code>
<b>Insertion</b>	Add new element	<code>arr[pos] = value;</code>
<b>Deletion</b>	Remove element	Shift elements
<b>Searching</b>	Find element	Linear/binary search
<b>Sorting</b>	Arrange elements	Bubble/Selection sort

#### Mnemonic

“DITIDSS: Declare-Initialize-Traverse-Insert-Delete-Search-Sort”

OR

### Question 4(b) [4 marks]

Define pointer. Explain with example.

#### Solution

**Pointer:** Variable that stores the memory address of another variable.

**Table 16.** Pointer Concepts

Concept	Description	Syntax
<b>Declaration</b>	Create pointer	<code>int *ptr;</code>
<b>Address operator</b>	Get address	<code>&amp;variable</code>
<b>Dereferencing</b>	Access value at address	<code>*ptr</code>
<b>Assignment</b>	Store address in pointer	<code>ptr = &amp;variable;</code>

**Example:**

```

1  #include <stdio.h>
2
3  int main() {
4      int num = 10;
5      int *ptr;
6
7      ptr = &num; // Store address of num in ptr
8
9      printf("Value of num: %d\n", num);           // 10
10     printf("Address of num: %p\n", &num);        // Address of num
11     printf("Value of ptr: %p\n", ptr);           // Same address
12     printf("Value pointed by ptr: %d\n", *ptr);  // 10
13

```



```

14  *ptr = 20;    // Change value using pointer
15  printf("New value of num: %d\n", num);    // 20
16
17  return 0;
18  }

```

Diagram:

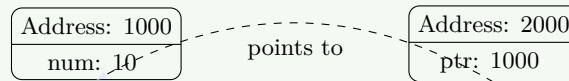


Figure 8. Pointer Memory Layout

### Mnemonic

“SAVD: Store Address, Value through Dereferencing”

OR

## Question 4(c) [7 marks]

Draw flowchart of while loop and explain with example.

### Solution

While Loop Syntax:

```

1  while (condition) {
2      // code block
3  }

```

Flowchart:

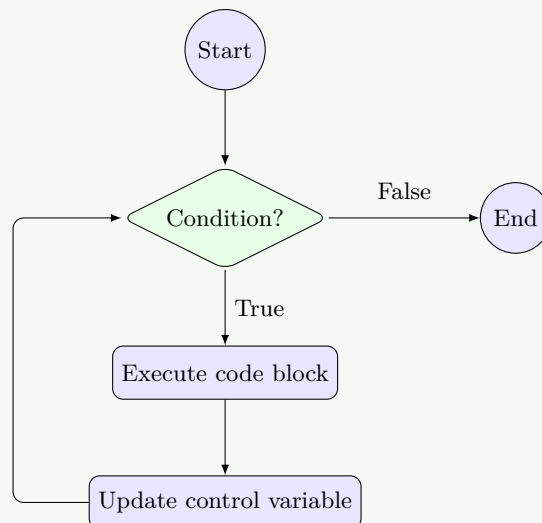


Figure 9. While Loop Flowchart

**While Loop Components:**

1. **Initialization:** Before the loop
2. **Condition:** Checked before each iteration
3. **Body:** Executed if condition is true
4. **Update:** Must be done inside the body

**Example:**

```

1  #include <stdio.h>
2
3  int main() {
4      int i = 1;
5
6      while (i <= 5) {
7          printf("%d ", i);
8          i++;
9      }
10     // Output: 1 2 3 4 5
11
12     return 0;
13 }

```

**Execution Flow:**

1. Initialize i = 1 (before loop)
2. Check condition (1 <= 5) - True
3. Execute body - Print 1
4. Update i to 2
5. Check condition (2 <= 5) - True
6. And so on until i becomes 6

**Mnemonic**

“CHECK-UPDATE: CHECK before entering, UPDATE before repeating”

**Question 5(a) [3 marks]**

State the use of following functions. (1) strcat() (2) strlen() (3) strcpy()

**Solution**

Table 17. String Functions

Function	Purpose	Syntax	Example
strcat()	Concatenates strings	strcat(dest, src)	"Hello" + "World" → "HelloWorld"
strlen()	Returns string length	strlen(str)	"Hello" → 5
strcpy()	Copies string	strcpy(dest, src)	src → dest

**Code Example:**

```

1  #include <string.h>
2
3  char str1[20] = "Hello";
4  char str2[20] = "World";
5  char str3[20];
6
7  strcat(str1, str2);    // str1 becomes "HelloWorld"
8  int len = strlen(str1); // len becomes 10
9  strcpy(str3, str1);    // str3 becomes "HelloWorld"

```

**Mnemonic**

“CLS: Concatenate-Length-Source copy”

## Question 5(b) [4 marks]

Build a structure to store book information: `book_no`, `book_title`, `book_author`, `book_price`.

### Solution

Listing 6. Book Structure

```

1  #include <stdio.h>
2  #include <string.h>
3
4  struct Book {
5      int book_no;
6      char book_title[50];
7      char book_author[30];
8      float book_price;
9  };
10
11 int main() {
12     struct Book book1;
13
14     // Assign values
15     book1.book_no = 101;
16     strcpy(book1.book_title, "Programming in C");
17     strcpy(book1.book_author, "Dennis Ritchie");
18     book1.book_price = 450.75;
19
20     // Display book information
21     printf("Book No: %d\n", book1.book_no);
22     printf("Title: %s\n", book1.book_title);
23     printf("Author: %s\n", book1.book_author);
24     printf("Price: %.2f\n", book1.book_price);
25
26     return 0;
27 }
```

Structure Memory Layout:

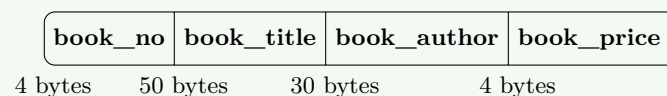


Figure 10. Structure Memory Representation

### Mnemonic

“NTAP: Number-Title-Author-Price”

## Question 5(c) [7 marks]

Explain array and array initialization. Give example.

### Solution

**Array:** Collection of same data type elements stored at contiguous memory locations.

**Array Initialization Methods:**

Table 18. Initialization Methods

Method	Syntax	Example
At declaration	type name[size] = {vals};	int arr[5] = {10, 20, 30, 40, 50};
Partial	type name[size] = {vals};	int arr[5] = {10, 20}; // Rest 0
All zeros	type name[size] = {0};	int arr[5] = {0};
Element wise	name[idx] = val;	arr[0] = 10; arr[1] = 20;
Size inference	type name[] = {vals};	int arr[] = {10, 20}; // Size 2

Example:

```
1 #include <stdio.h>
2
3 int main() {
4     // Method 1: Complete initialization
5     int arr1[5] = {10, 20, 30, 40, 50};
6
7     // Method 2: Partial initialization
8     int arr2[5] = {10, 20}; // Remaining are 0
9
10    // Method 4: Size inference
11    int arr4[] = {1, 2, 3, 4, 5};
12
13    // Accessing elements
14    printf("arr1[2] = %d\n", arr1[2]); // Output: 30
15
16    return 0;
17 }
```

Memory Representation:

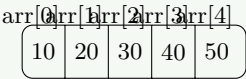


Figure 11. Array Memory Layout

Mnemonic

“CAPES: Complete, Automatic, Partial, Element, Size-inferred”

OR

Question 5(a) [3 marks]

Compare array and structure with example.

Solution

Table 19. Array vs Structure

Feature	Array	Structure
Data type	Same type elements	Different type elements
Access	Using index ( <code>arr[i]</code> )	Using dot operator ( <code>s.member</code> )
Memory	Contiguous, fixed size	Contiguous, may have padding
Assignment	Element by element	Direct with compatible structures
Purpose	Collection of similar items	Group of related data

**Example:**

```

1 // Array
2 int marks[5] = {85, 90, 78, 92, 88};
3 printf("%d", marks[2]);
4
5 // Structure
6 struct Student { int roll; char name[20]; };
7 struct Student s1 = {101, "Raj"};
8 printf("%s", s1.name);

```

**Mnemonic**

“DAMPA: Datatype-Access-Memory-Purpose-Assignment”

OR

**Question 5(b) [4 marks]**

Define User Defined Function. Explain with example.

**Solution**

**User Defined Function:** Block of code written by programmer to perform specific task, which can be called multiple times.

**Table 20.** Function Components

Component	Description	Example
Return type	Data type returned	int, float, void
Function name	Unique identifier	sum, findMax
Parameters	Input data	(int a, int b)
Body	Set of statements	{ return a+b; }

**Example:**

```

1 #include <stdio.h>
2
3 // Function declaration
4 int sum(int a, int b);
5
6 int main() {
7     int num1 = 10, num2 = 20, result;
8
9     // Function call
10    result = sum(num1, num2);
11
12    printf("Sum = %d\n", result);

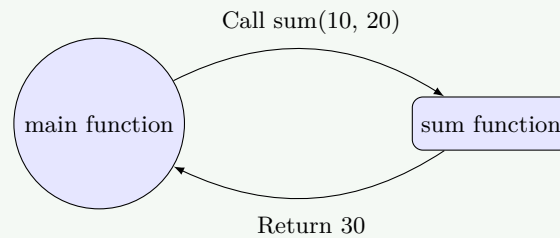
```

```

13     return 0;
14 }
15
16 // Function definition
17 int sum(int a, int b) {
18     return a + b;
19 }
20

```

**Function Flow:**



**Figure 12.** Function Call Flow

### Mnemonic

“DPCR: Declaration-Parameters-Call-Return”

OR

## Question 5(c) [7 marks]

Develop a C program to find maximum and minimum element of array.

### Solution

**Listing 7.** Min Max Array

```

1  #include <stdio.h>
2
3  int main() {
4      int arr[100], n, i;
5      int max, min;
6
7      printf("Enter number of elements: ");
8      scanf("%d", &n);
9
10     printf("Enter %d integers: ", n);
11     for(i = 0; i < n; i++) {
12         scanf("%d", &arr[i]);
13     }
14
15     // Initialize max and min
16     max = min = arr[0];
17
18     // Find max and min
19     for(i = 1; i < n; i++) {
20         if(arr[i] > max)
21             max = arr[i];
22         if(arr[i] < min)
23             min = arr[i];

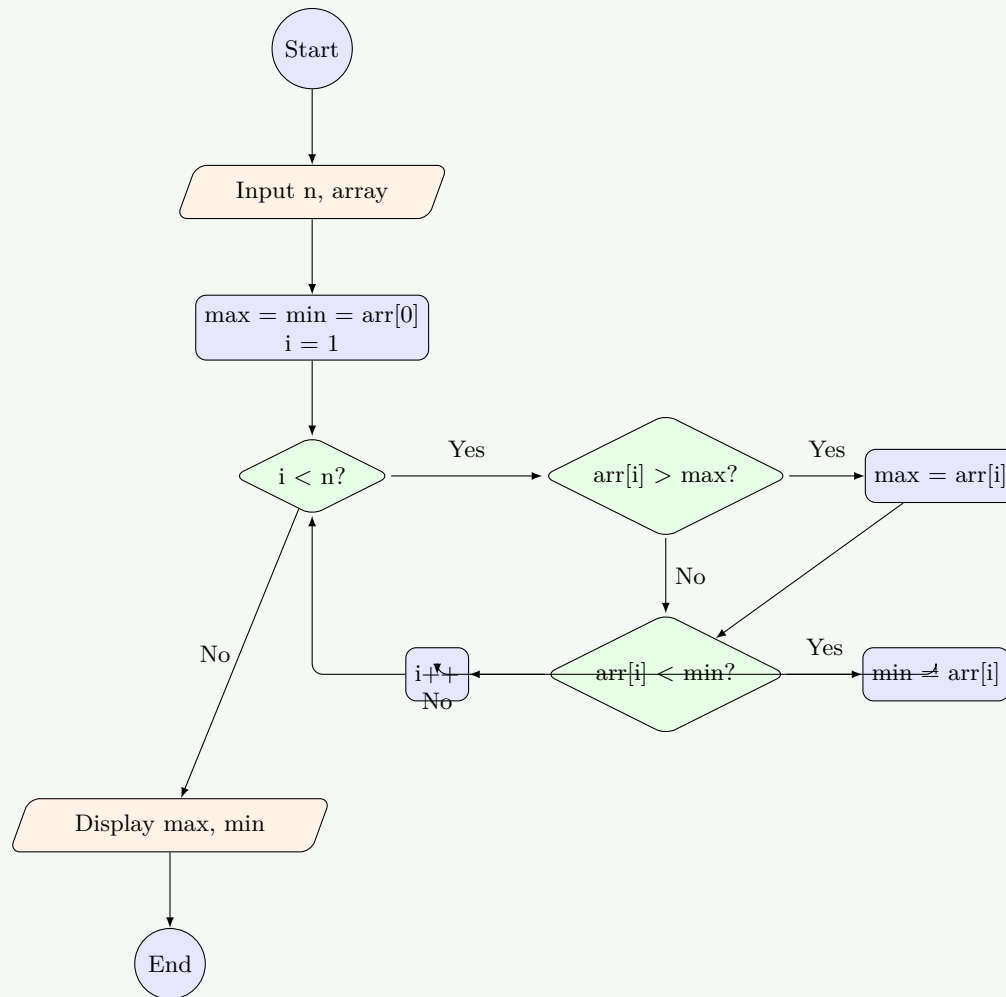
```

```

24     }
25
26     printf("Maximum: %d\n", max);
27     printf("Minimum: %d\n", min);
28
29     return 0;
30 }

```

**Flowchart:**



**Figure 13.** Flowchart for Min Max

### Mnemonic

“FILLS: First Initialize, Loop through, Look for Small/large”