

Programming in C (4331105) - Winter 2023 Solution

Milav Dabgar

May 21, 2024

Question 1 [a marks]

3 Define algorithm and write an algorithm to find area of circle.

Solution

Answer: An algorithm is a step-by-step procedure or set of rules for solving a specific problem or accomplishing a particular task.

Algorithm to find area of circle:

1. Start
2. Input radius (r) of the circle
3. Calculate area = $\pi \times r^2$
4. Display the area
5. Stop

Mnemonic

"Start, Read, Calculate, Display, Stop"

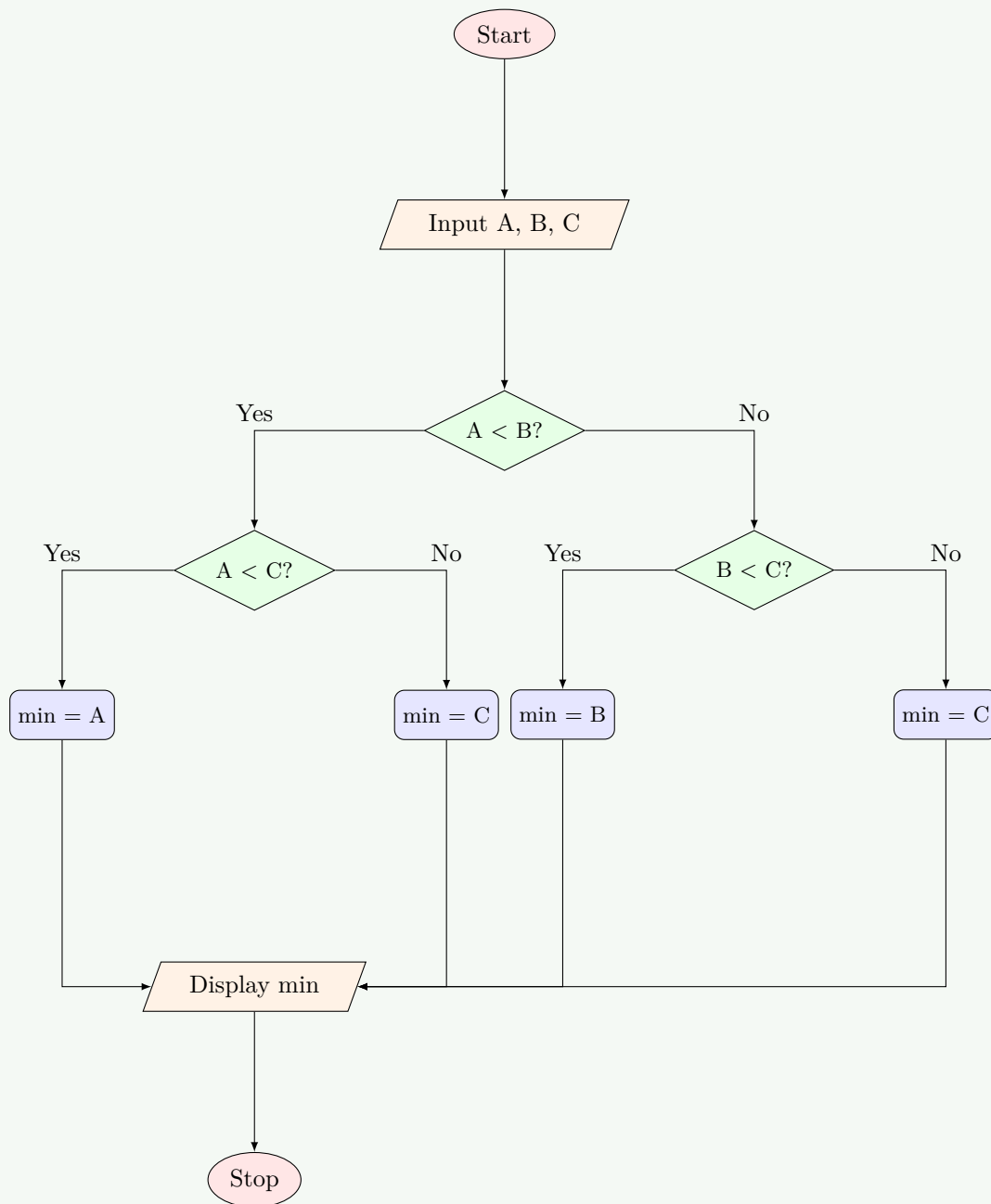
Question 1 [b marks]

4 Define flowchart and draw a flowchart to find minimum of three numbers.

Solution

Answer: A flowchart is a visual representation of an algorithm using standardized symbols and shapes connected by arrows to show the sequence of steps.

Flowchart to find minimum of three numbers:



- **Comparison Strategy:** First compare A and B, then compare with C
- **Branching Logic:** Use if-else structure to find smallest value

Mnemonic

"Compare pairs, find the rare small value everywhere"

Question 1 [c marks]

7 Write a program to calculate simple interest using below equation. $I = \frac{PRN}{100}$ Where P=Principle amount, R=Rate of interest and N=Period.

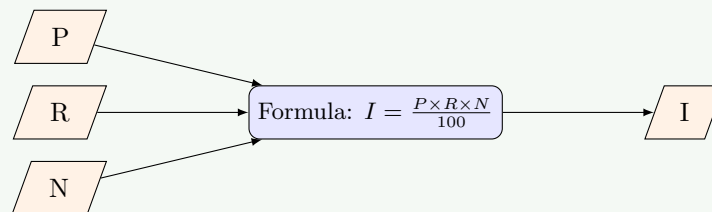
Solution

Answer:

```

1  #include <stdio.h>
2
3  int main() {
4      float P, R, N, I;
5
6      // Input principal amount, rate of interest and time period
7      printf("Enter Principal amount: ");
8      scanf("%f", &P);
9
10     printf("Enter Rate of interest: ");
11     scanf("%f", &R);
12
13     printf("Enter Time period (in years): ");
14     scanf("%f", &N);
15
16     // Calculate Simple Interest
17     I = (P * R * N) / 100;
18
19     // Display the result
20     printf("Simple Interest = %.2f\n", I);
21
22     return 0;
23 }
```

Diagram:



- **Floating-point variables:** Store decimal values for precision
- **User interaction:** Clear prompts for input
- **Result formatting:** %.2f displays two decimal places

Mnemonic

"Principal, Rate and Number, divided by Hundred gives Interest"

OR

Question 1 [c marks]

7 Write a program to read radius(R) and height(H) from keyboard and print calculated the volume(V) of cylinder using $V = \pi R^2 H$

Solution

Answer:

```

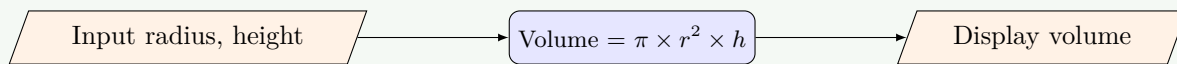
1  #include <stdio.h>
2
3  int main() {
```

```

4  float radius, height, volume;
5  const float PI = 3.14159;
6
7  // Input radius and height
8  printf("Enter radius of cylinder: ");
9  scanf("%f", &radius);
10
11 printf("Enter height of cylinder: ");
12 scanf("%f", &height);
13
14 // Calculate volume of cylinder
15 volume = PI * radius * radius * height;
16
17 // Display the result
18 printf("Volume of cylinder = %.2f\n", volume);
19
20 return 0;
21 }

```

Diagram:



- **Constants:** PI defined as constant for clarity
- **Formula:** Use R^2 by multiplying radius twice
- **Input validation:** Assumes positive values for radius and height

Mnemonic

"Radius squared times height times Pi, gives cylinder volume, don't ask why"

Question 2 [a marks]

3 List out different operators supported in C programming language.

Solution

Answer:

Category	Operators
Arithmetic	+, -, *, /, % (addition, subtraction, multiplication, division, modulus)
Relational	==, !=, >, <, >=, <= (equal, not equal, greater than, less than, greater than or equal to, less than or equal to)
Logical	&&, , ! (AND, OR, NOT)
Assignment	=, +=, -=, *=, /=, %= (assign, plus-assign, minus-assign, etc.)
Increment/Decrement	++, -- (increment, decrement)
Bitwise	&, , ^, ~, <<, >> (AND, OR, XOR, complement, left shift, right shift)
Conditional	? : (ternary operator)
Special	sizeof(), &, *, ->, . (size, address, pointer, structure)

Mnemonic

"ARABIA CS" (Arithmetic, Relational, Assignment, Bitwise, Increment, Assignment, Conditional, Special)

Question 2 [b marks]

4 Explain Relational operator and Increment/Decrement operator with example.

Solution

Answer:

Operator Type	Description	Example	Output
Relational	Compare two values to test the relationship between them	int a = 5, b = 10; printf("%d", a < b);	1 (true)
	Equal to (==)	printf("%d", 5 == 5);	1 (true)
	Not equal to (!=)	printf("%d", 5 != 10);	1 (true)
	Greater/Less than	printf("%d %d", 5 > 3, 5 < 3);	1 0
Increment	Increases value by 1. Pre-increment (++x): increment then use. Post-increment (x++): use then increment	int x = 5; printf("%d ", ++x); printf("%d", x);	6 6
Decrement	Decreases value by 1. Pre-decrement (--x): decrement then use. Post-decrement (x--): use then decrement	int y = 5; printf("%d ", y--); printf("%d", y);	5 4

- **Relational operators:** Return 1 (true) or 0 (false)
- **Increment/Decrement:** Changes variable value and returns a value

Mnemonic

"Relational tells if TRUE or LIE, Increment/Decrement makes values rise or DIE"

Question 2 [c marks]

7 Write a program to print sum and average of 1 to 100.

Solution

Answer:

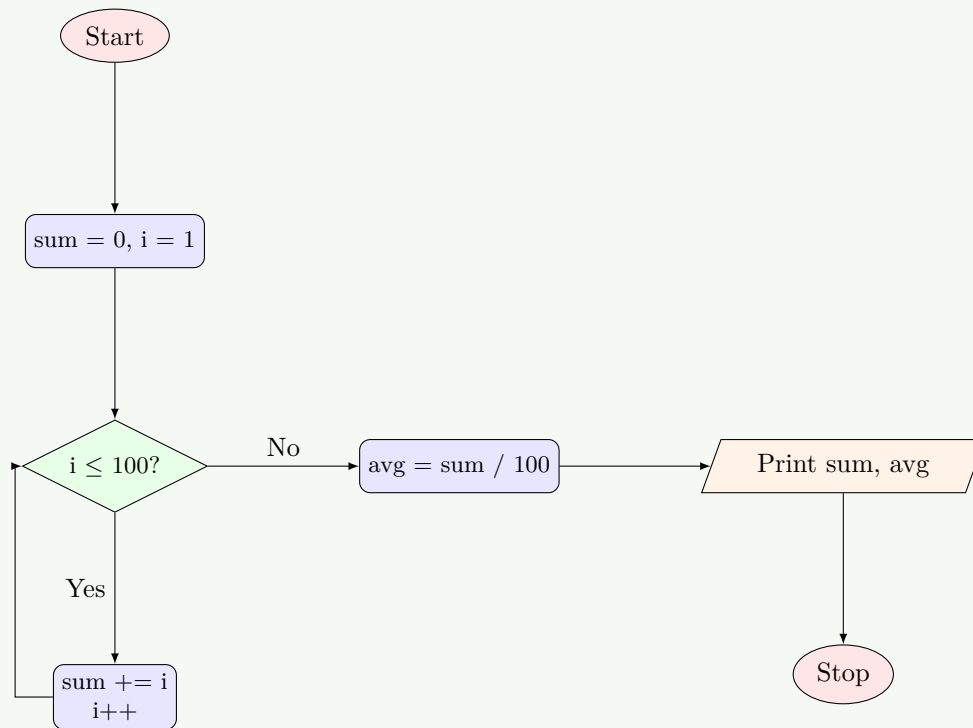
```

1  #include <stdio.h>
2
3  int main() {
4      int i, sum = 0;
5      float average;
6
7      // Calculate sum of numbers from 1 to 100
8      for(i = 1; i <= 100; i++) {
9          sum += i;
10     }
11
12     // Calculate average
13     average = (float)sum / 100;
14
15     // Display the results
16     printf("Sum of numbers from 1 to 100 = %d\n", sum);
17     printf("Average of numbers from 1 to 100 = %.2f\n", average);
18
19     return 0;

```

20 }

Diagram:



- **Loop counter:** Variable *i* tracks numbers 1 to 100
- **Sum calculation:** Accumulates values in *sum* variable
- **Type casting:** (float) converts *sum* to floating-point for accurate division

Mnemonic

"Sum One to Hundred, then Divide for Average"

OR

Question 2 [a marks]

3 State the difference between `gets(S)` and `scanf("%s",S)` where *S* is string.

Solution**Answer:**

Feature	<code>gets(S)</code>	<code>scanf("%s",S)</code>
Input termination	Reads until newline character (<code>\n</code>)	Reads until whitespace (space, tab, newline)
Whitespace handling	Can read string with spaces	Stops reading at first whitespace
Buffer overflow	No bounds checking (unsafe)	No bounds checking (unsafe)
Return value	Returns <i>S</i> on success, NULL on error	Returns number of items successfully read
Replacement	<code>fgets()</code> is safer alternative	<code>scanf("%ns",S)</code> with width limit is safer

- **Safety concern:** Both functions can cause buffer overflow
- **Practical usage:** `gets()` for full lines, `scanf()` for single words

Mnemonic

"gets Gets Everything Till newline, scanf Stops Catching After Finding whitespace"

OR

Question 2 [b marks]

4 Explain Logical operator and Assignment operator with example.

Solution

Answer:

Operator Type	Description	Example	Out-put
Logical	Perform logical operations on conditions	<code>int a = 5, b = 10;</code>	
	Logical AND (&&)	<code>printf("%d", (a > 0) && (b > 0));</code>	1 (true)
	Logical OR ()	<code>printf("%d", (a > 10) (b > 5));</code>	1 (true)
	Logical NOT (!)	<code>printf("%d", !(a == b));</code>	1 (true)
Assignment	Assign values to variables	<code>int x = 10;</code>	x = 10
	Simple assignment (=)	<code>x = 20;</code>	x = 20
	Add and assign (+=)	<code>x += 5;</code>	x = 25
	Subtract and assign (-=)	<code>x -= 10;</code>	x = 15
	Multiply and assign (*=)	<code>x *= 2;</code>	x = 30
	Divide and assign (/=)	<code>x /= 3;</code>	x = 10

- **Logical operators:** Used in decision making
- **Short-circuit evaluation:** && and || evaluate only what's necessary
- **Compound assignment:** Combines operation and assignment

Mnemonic

"AND needs all TRUE, OR needs just one; Assignment takes right, puts it on the left throne"

OR

Question 2 [c marks]

7 Write a program to print all the integers between given two floating point numbers.

Solution

Answer:

```

1  #include <stdio.h>
2  #include <math.h>
3
4  int main() {
5      float num1, num2;
6      int start, end, i;

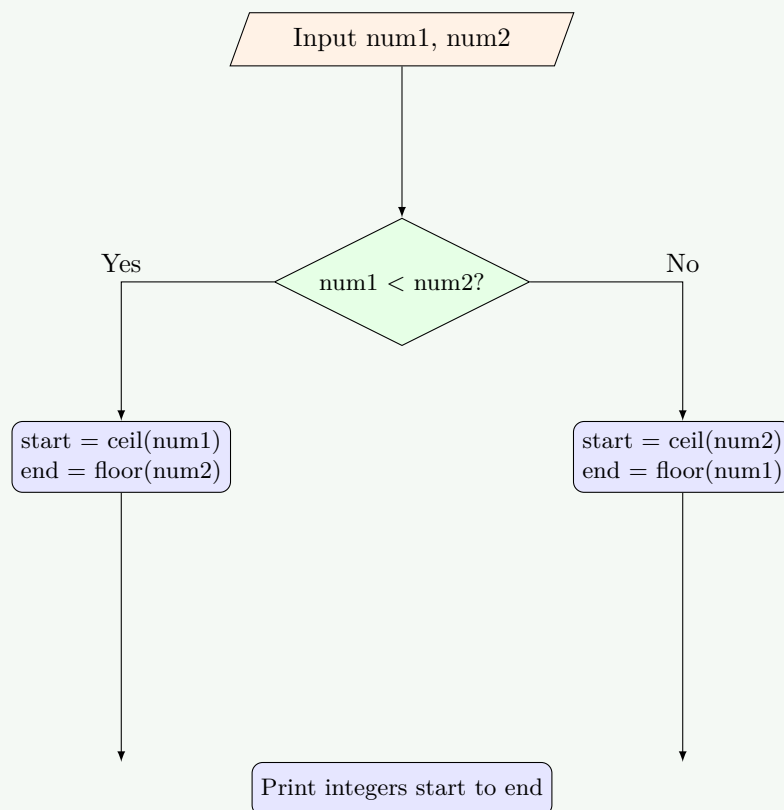
```

```

7
8 // Input two floating point numbers
9 printf("Enter first floating point number: ");
10 scanf("%f", &num1);
11
12 printf("Enter second floating point number: ");
13 scanf("%f", &num2);
14
15 // Find the ceil of smaller number and floor of larger number
16 if(num1 < num2) {
17     start = ceil(num1);
18     end = floor(num2);
19 } else {
20     start = ceil(num2);
21     end = floor(num1);
22 }
23
24 // Print all integers between the two numbers
25 printf("Integers between %.2f and %.2f are:\n", num1, num2);
26 for(i = start; i <= end; i++) {
27     printf("%d ", i);
28 }
29 printf("\n");
30
31 return 0;
32 }

```

Diagram:



- **Math functions:** `ceil()` rounds up, `floor()` rounds down
- **Range determination:** Works regardless of input order
- **Integer extraction:** Only prints whole numbers between floats

Mnemonic

"Ceiling the small, flooring the big, then print every Integer in between"

Question 3 [a marks]

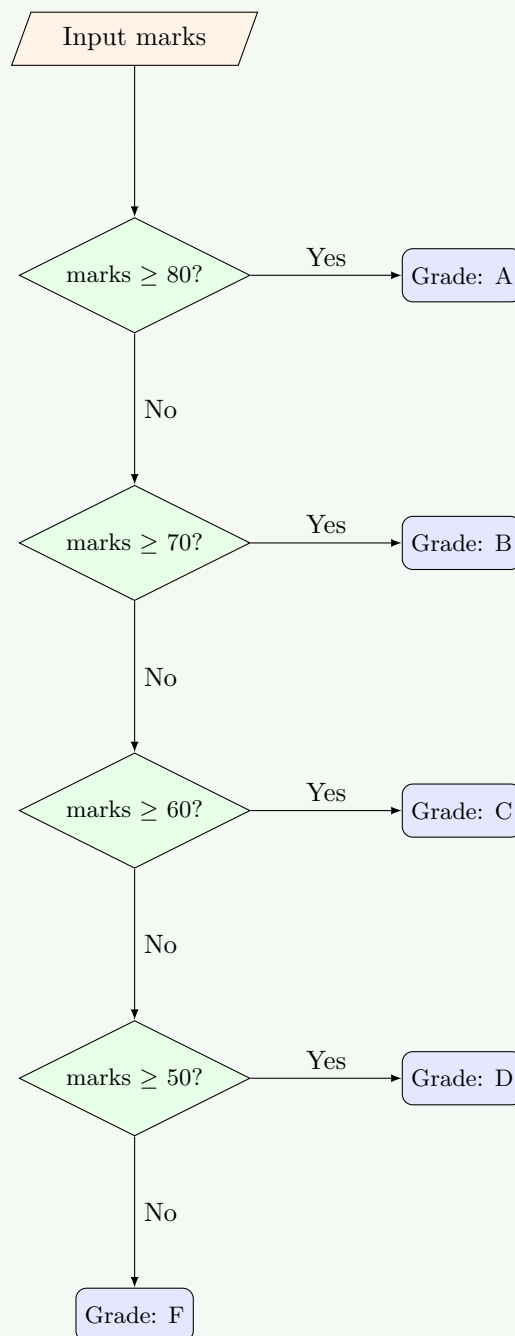
3 Explain multiple if-else statement with example.

Solution

Answer: Multiple if-else statements allow testing several conditions in sequence, where each condition is checked only if the previous conditions are false.

```
1  #include <stdio.h>
2
3  int main() {
4      int marks;
5
6      printf("Enter marks (0-100): ");
7      scanf("%d", &marks);
8
9      if(marks >= 80) {
10         printf("Grade: A\n");
11     } else if(marks >= 70) {
12         printf("Grade: B\n");
13     } else if(marks >= 60) {
14         printf("Grade: C\n");
15     } else if(marks >= 50) {
16         printf("Grade: D\n");
17     } else {
18         printf("Grade: F\n");
19     }
20
21     return 0;
22 }
```

Diagram:



- **Sequential testing:** Only one block executes
- **Efficiency:** Stops checking after finding true condition

Mnemonic

"If this THEN that, ELSE IF another THEN something else"

Question 3 [b marks]

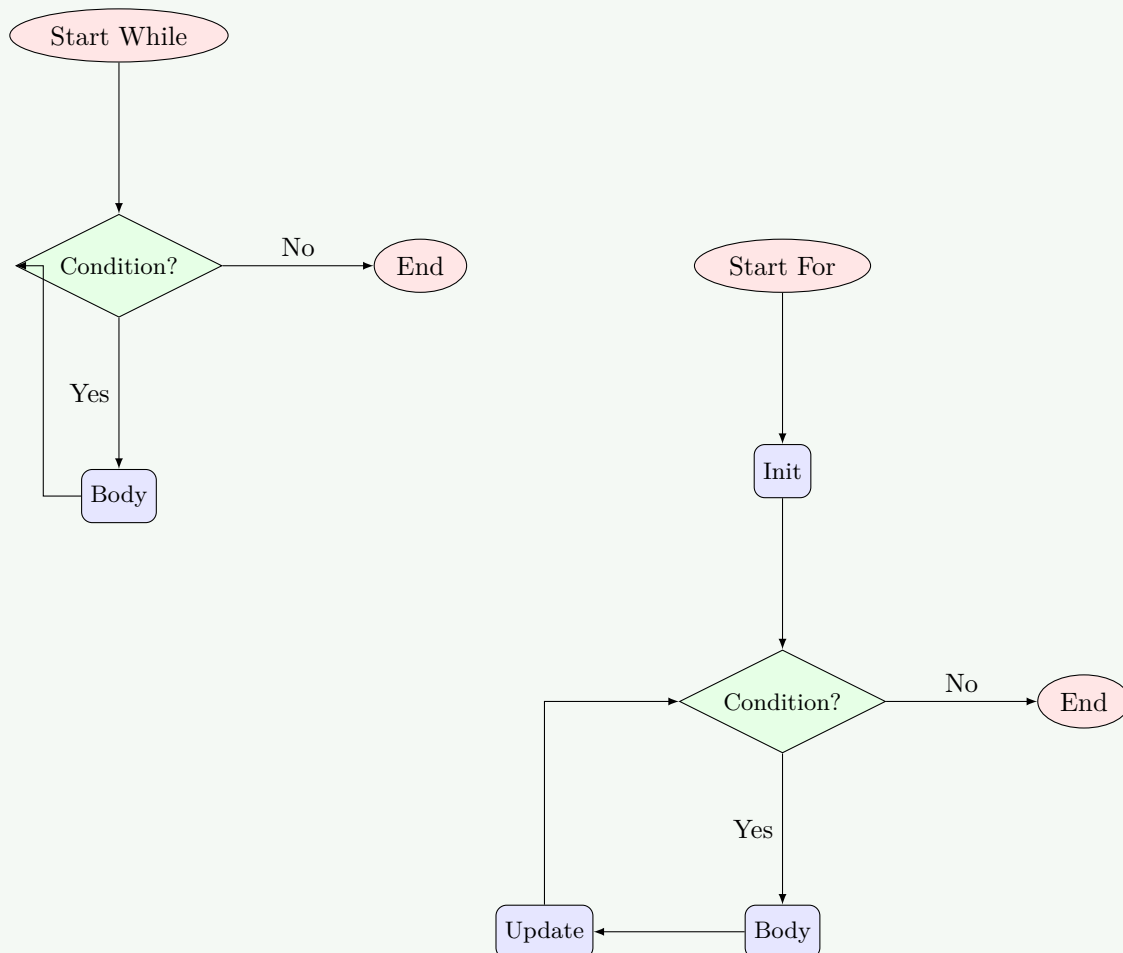
4 State the working of while loop and for loop.

Solution

Answer:

Loop Type	Working	Syntax	Use Cases
while loop	<ol style="list-style-type: none"> 1. Test condition 2. If true, execute body 3. Repeat steps 1-2 until condition is false 	<pre>while(condition) { // statements }</pre>	When number of iterations is unknown beforehand
for loop	<ol style="list-style-type: none"> 1. Execute initialization once 2. Test condition 3. If true, execute body 4. Execute update statement 5. Repeat steps 2-4 until condition is false 	<pre>for(init; cond; update) { // statements }</pre>	When number of iterations is known beforehand

Comparison:



- **Entry control:** Both check condition before execution
- **Components:** for loop combines initialization, condition, and update

Mnemonic

"WHILE checks THEN acts, FOR initializes CHECKS acts UPDATES"

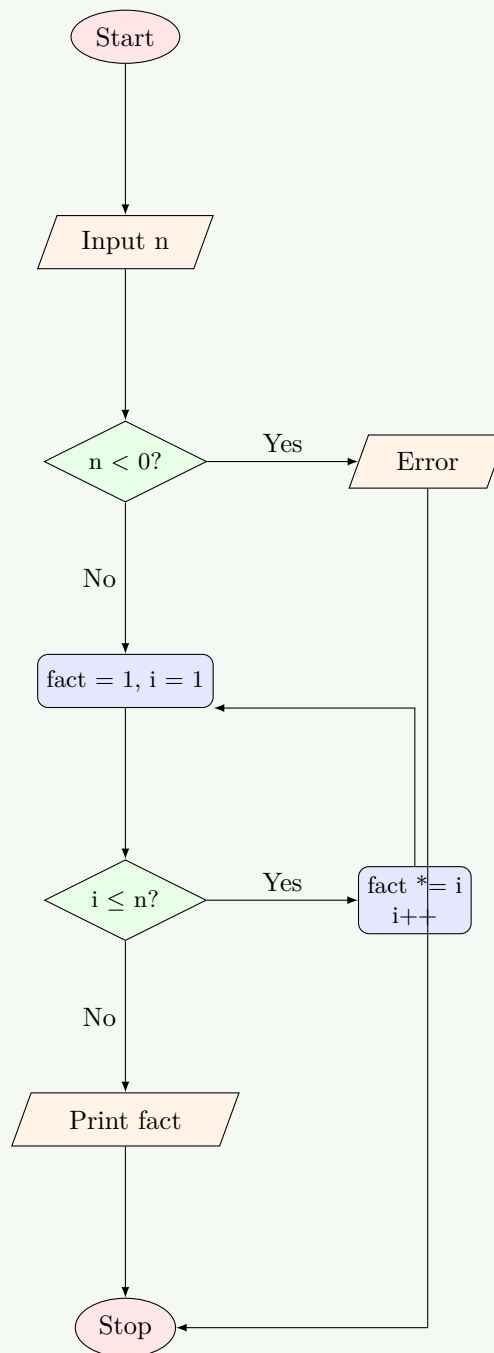
Question 3 [c marks]

7 Write a program to find factorial of a given number.

Solution**Answer:**

```
1  #include <stdio.h>
2
3  int main() {
4      int num, i;
5      unsigned long long factorial = 1;
6
7      // Input a number
8      printf("Enter a positive integer: ");
9      scanf("%d", &num);
10
11     // Check if the number is negative
12     if(num < 0) {
13         printf("Error: Factorial is not defined for negative numbers.\n");
14     } else {
15         // Calculate factorial
16         for(i = 1; i <= num; i++) {
17             factorial *= i;
18         }
19
20         printf("Factorial of %d = %llu\n", num, factorial);
21     }
22
23     return 0;
24 }
```

Diagram:



- **Data type:** unsigned long long for large factorials
- **Error handling:** Checks for negative input
- **Loop implementation:** Multiply successive integers

Mnemonic

"Factorial Formula: Multiply From One to Number"

OR

Question 3 [a marks]

3 Explain the working of switch-case statement with example.

Solution

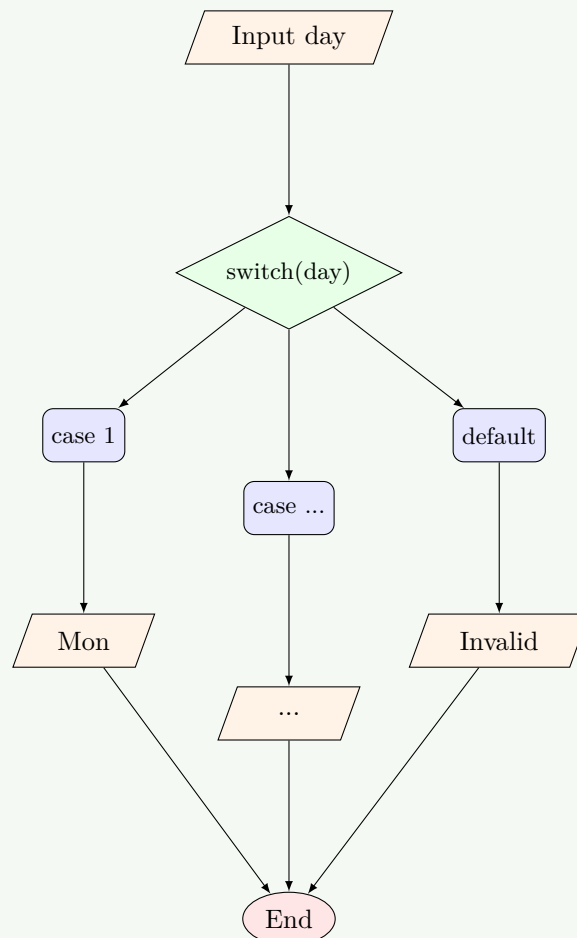
Answer: The switch-case statement is a multi-way decision maker that tests the value of an expression against various case values and executes the matching case block.

```

1  #include <stdio.h>
2
3  int main() {
4      int day;
5
6      printf("Enter day number (1-7): ");
7      scanf("%d", &day);
8
9      switch(day) {
10         case 1: printf("Monday\n"); break;
11         case 2: printf("Tuesday\n"); break;
12         case 3: printf("Wednesday\n"); break;
13         case 4: printf("Thursday\n"); break;
14         case 5: printf("Friday\n"); break;
15         case 6: printf("Saturday\n"); break;
16         case 7: printf("Sunday\n"); break;
17         default: printf("Invalid day number\n");
18     }
19
20     return 0;
21 }

```

Diagram:



- **Expression evaluation:** Only integer or character types

- **Case matching:** Executes matching case until break
- **Default case:** Executes when no case matches

Mnemonic

"SWITCH value, CASE match, BREAK out, DEFAULT rescue"

OR

Question 3 [b marks]

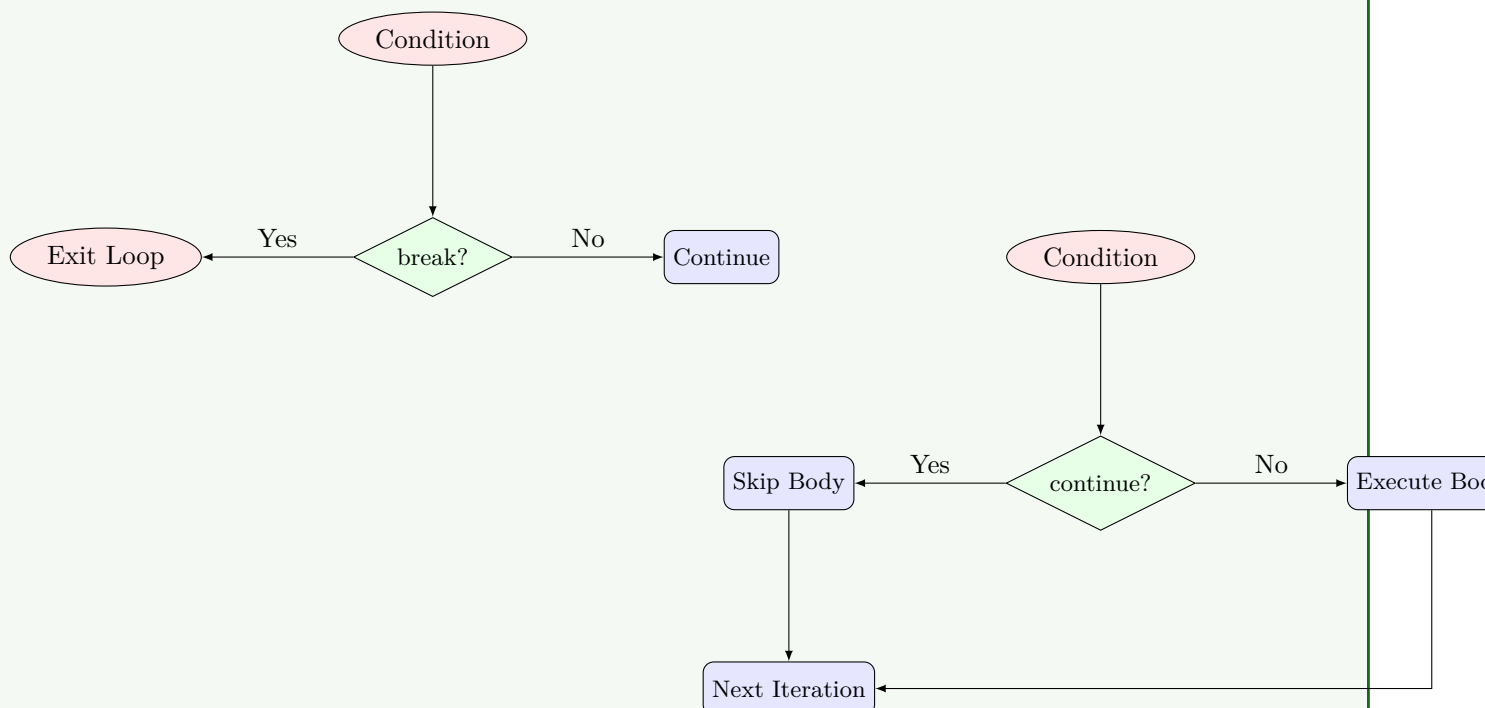
4 Define break and continue keyword.

Solution

Answer:

Key-word	Definition	Purpose	Example
break	Terminates the innermost loop or switch statement immediately	To exit a loop prematurely when a certain condition is met	<pre>for(i=1; i<=10; i++) { if(i == 5) break; printf("%d ", i); } // Output: 1 2 3 4</pre>
continue	Skips the rest of the current iteration and jumps to the next iteration of the loop	To skip specific iterations without terminating the loop	<pre>for(i=1; i<=10; i++) { if(i == 5) continue; printf("%d ", i); } // Output: 1 2 3 4 6 7 8 9 10</pre>

Behavioral Comparison:



- **Scope:** Both affect only the innermost loop
- **Control transfer:** break exits loop, continue jumps to next iteration

Mnemonic

"BREAK leaves the room, CONTINUE skips to the next dance move"

OR

Question 3 [c marks]

7 Write a program to read number of lines (n) from keyboard and print the triangle shown below.

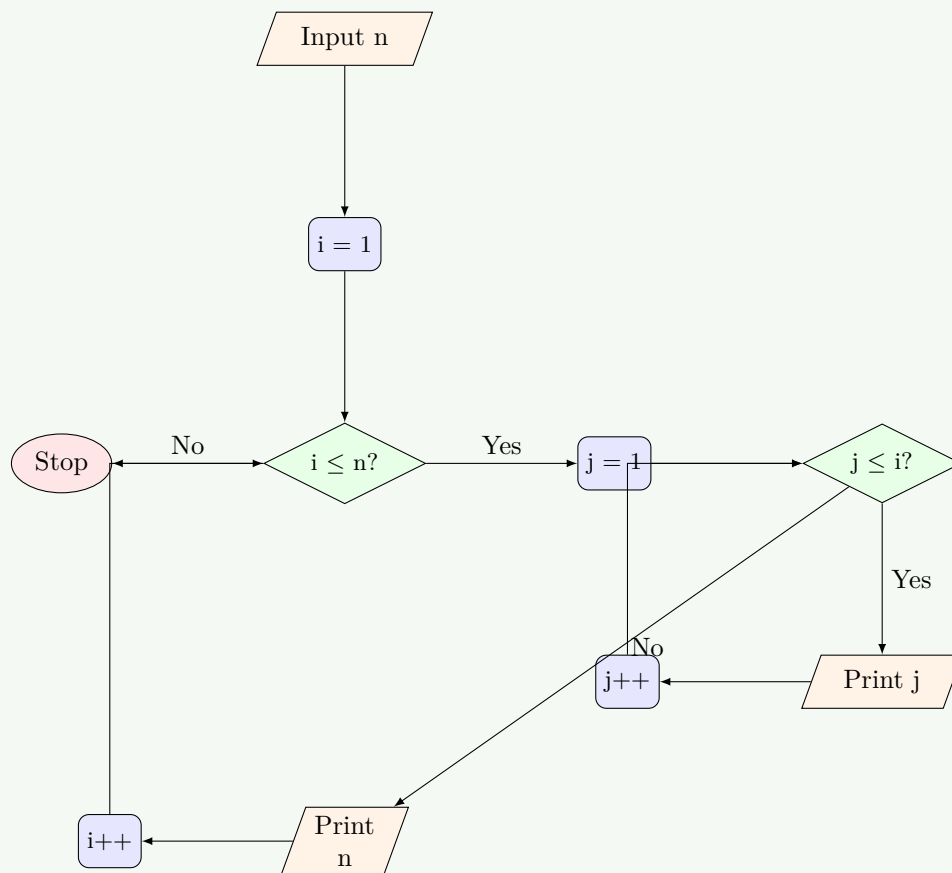
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Solution

Answer:

```
1  #include <stdio.h>
2
3  int main() {
4      int n, i, j;
5
6      // Input number of lines
7      printf("Enter number of lines: ");
8      scanf("%d", &n);
9
10     // Print the triangle pattern
11     for(i = 1; i <= n; i++) {
12         // Print numbers from 1 to i in each row
13         for(j = 1; j <= i; j++) {
14             printf("%d ", j);
15         }
16         printf("\n");
17     }
18
19     return 0;
20 }
```

Program Flow:



- **Nested loops:** Outer loop for rows, inner loop for columns
- **Pattern logic:** Row number determines how many numbers to print
- **Number sequence:** Each row prints 1 to row number

Mnemonic

"Rows decide COUNTER limit, COLumns print ONE to ROW"

Question 4 [a marks]

3 Explain nested if-else statement with example.

Solution

Answer: Nested if-else statements are if-else constructs placed inside another if or else block, allowing more complex conditional logic and multiple levels of decision making.

```

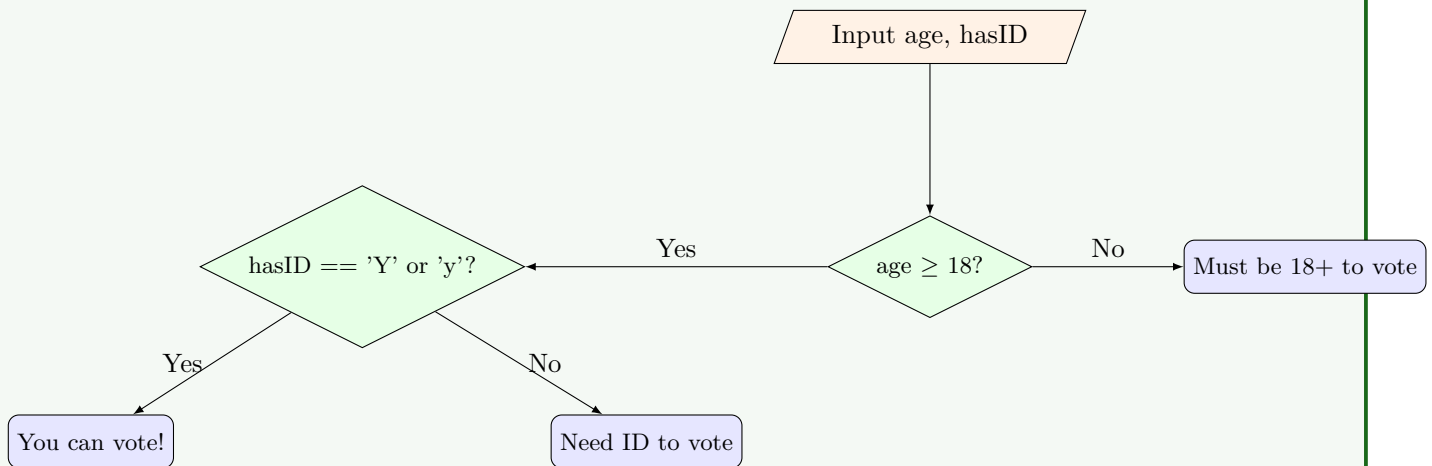
1  #include <stdio.h>
2
3  int main() {
4      int age;
5      char hasID;
6
7      printf("Enter age: ");
8      scanf("%d", &age);
9
10     printf("Do you have ID? (Y/N): ");
11     scanf(" %c", &hasID);
  
```

```

12
13     if(age >= 18) {
14         if(hasID == 'Y' || hasID == 'y') {
15             printf("You can vote!\n");
16         } else {
17             printf("You need ID to vote.\n");
18         }
19     } else {
20         printf("You must be 18 or older to vote.\n");
21     }
22
23     return 0;
24 }

```

Decision Tree:



- **Hierarchical conditions:** Evaluates conditions in layers
- **Indentation:** Improves readability of nested structures
- **Multi-factor decisions:** Combines multiple criteria

Mnemonic

"If INSIDE if, check DEEPER conditions"

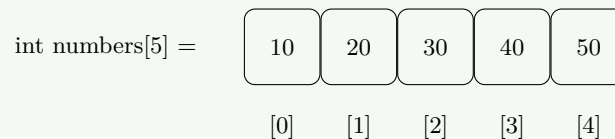
Question 4 [b marks]

4 Describe initialization of one-dimensional array.

Solution

Answer:

Initialization Method	Syntax	Example	Description
Declaration with size	<code>type name[size];</code>	<code>int marks[5];</code>	Creates array with specified size, elements have garbage values
Declaration with initialization	<code>type name[size] = {values};</code>	<code>int ages[4] = {21, 19, 25, 32};</code>	Creates and initializes array with specific values
Partial initialization	<code>type name[size] = {values};</code>	<code>int nums[5] = {1, 2};</code>	Initializes first elements, rest become zero
Size inference	<code>type name[] = {values};</code>	<code>int scores[] = {95, 88, 72};</code>	Size determined by number of initializers
Individual element	<code>name[index] = value;</code>	<code>marks[0] = 85;</code>	Assigns value to specific element

Array Visualization:

- **Zero-indexing:** First element at index 0
- **Contiguous memory:** Elements stored sequentially
- **Size limitation:** Size must be known at compile time

Mnemonic

"Declare SIZE first, then FILL with values or let COMPILER COUNT"

Question 4 [c marks]

7 Define Array and write a program to reverse a string.

Solution

Answer: An array is a collection of similar data items stored at contiguous memory locations and accessed using a common name.

```

1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char str[100], reversed[100];
6      int i, j, length;
7
8      // Input a string
9      printf("Enter a string: ");
10     gets(str);
11
12     // Find the length of string
13     length = strlen(str);
14
15     // Reverse the string
16     for(i = length - 1, j = 0; i >= 0; i--, j++) {
17         reversed[j] = str[i];
18     }
19

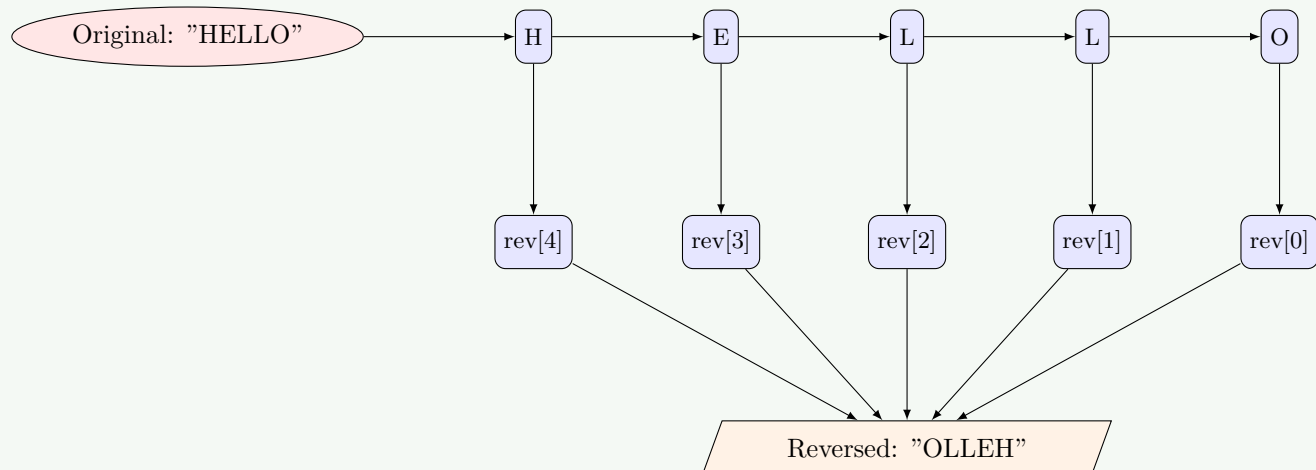
```

```

20 // Add null terminator
21 reversed[j] = '\0';
22
23 // Display the reversed string
24 printf("Reversed string: %s\n", reversed);
25
26 return 0;
27 }

```

Algorithm Visualization:



- **Character array:** Stores string with null terminator
- **Two-pointer technique:** One for original, one for reversed
- **Zero-based indexing:** Arrays start at index 0

Mnemonic

"Start from END, place at BEGIN, stop at ZERO"

OR

Question 4 [a marks]

3 Explain do while loop with example

Solution

Answer: The do-while loop is an exit-controlled loop that executes the loop body at least once before checking the condition.

```

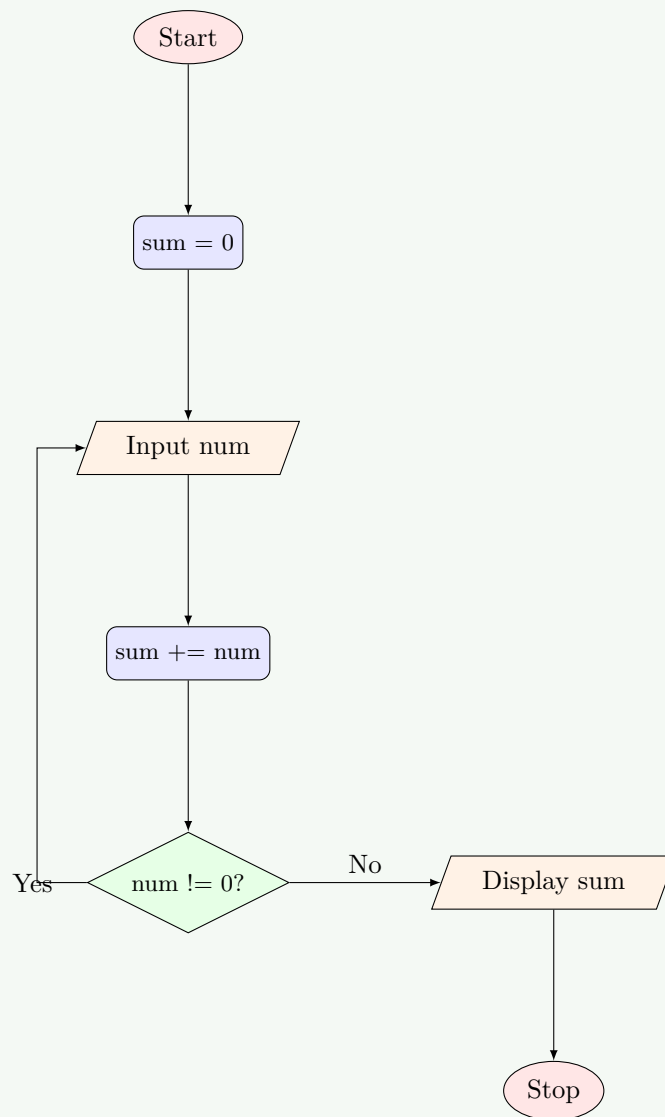
1 #include <stdio.h>
2
3 int main() {
4     int num, sum = 0;
5
6     do {
7         printf("Enter a number (0 to stop): ");
8         scanf("%d", &num);
9         sum += num;
10    } while(num != 0);
11
12    printf("Sum of all entered numbers: %d\n", sum);

```

```

13 |
14 |     return 0;
15 | }

```

Loop Execution Flow:

- **Execution order:** Body first, condition check later
- **Guaranteed execution:** Loop body always executes at least once
- **Termination:** Condition evaluated at bottom of loop

Mnemonic

"DO first, ask questions WHILE later"

OR

Question 4 [b marks]

4 Define pointer and describe pointer with example.

Solution

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

Pointer Concept	Description	Example
Declaration	Data_type *pointer_name;	int *ptr;
Initialization	Assign address of a variable	int num = 10; int *ptr = #
Dereference	Access the value at the address	printf("%d", *ptr); // Prints 10
Address operator	Gets address of a variable	printf("%p", &num); // Prints address
Null pointer	Pointer that points to nothing	int *ptr = NULL;

Pointer Visualization:



- **Indirect access:** Access variables through their addresses
- **Memory manipulation:** Direct memory access for efficiency
- **Dynamic memory:** Enables allocation/deallocation during runtime

Mnemonic

"Pointers POINT to ADDRESS, STARS dereference to VALUES"

OR

Question 4 [c marks]

7 Define pointer and write a program to exchange two integers using pointer arguments.

Solution

Answer: A pointer is a variable that contains the memory address of another variable, allowing indirect access and manipulation of data.

```

1  #include <stdio.h>
2
3  // Function to swap two integers using pointers
4  void swap(int *a, int *b) {
5      int temp = *a;
6      *a = *b;
7      *b = temp;
8  }
9
10 int main() {
11     int num1, num2;
12
13     // Input two integers
14     printf("Enter first number: ");
15     scanf("%d", &num1);
16
17     printf("Enter second number: ");
18     scanf("%d", &num2);
19
20     printf("Before swapping: num1 = %d, num2 = %d\n", num1, num2);
21

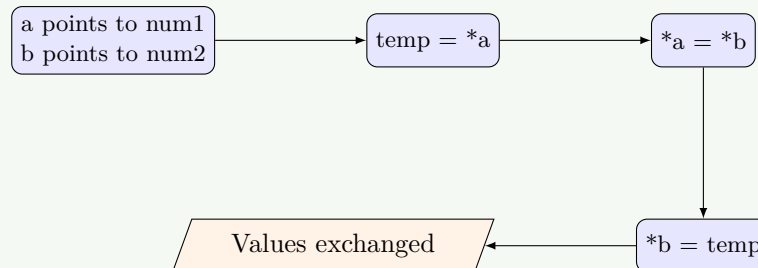
```

```

22 // Call swap function with addresses of num1 and num2
23 swap(&num1, &num2);
24
25 printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);
26
27 return 0;
28 }

```

Swap Process Visualization:



- **Pass by reference:** Pointers allow functions to modify original variables
- **Temporary variable:** Required for swapping without data loss
- **Function parameter:** Pointer arguments pass addresses

Mnemonic

"Grab by ADDRESS, change the CONTENT, without being PRESENT"

Question 5 [a marks]

3 Write a program to find the numbers which are divisible by 7 in between the numbers 50 and 500.

Solution

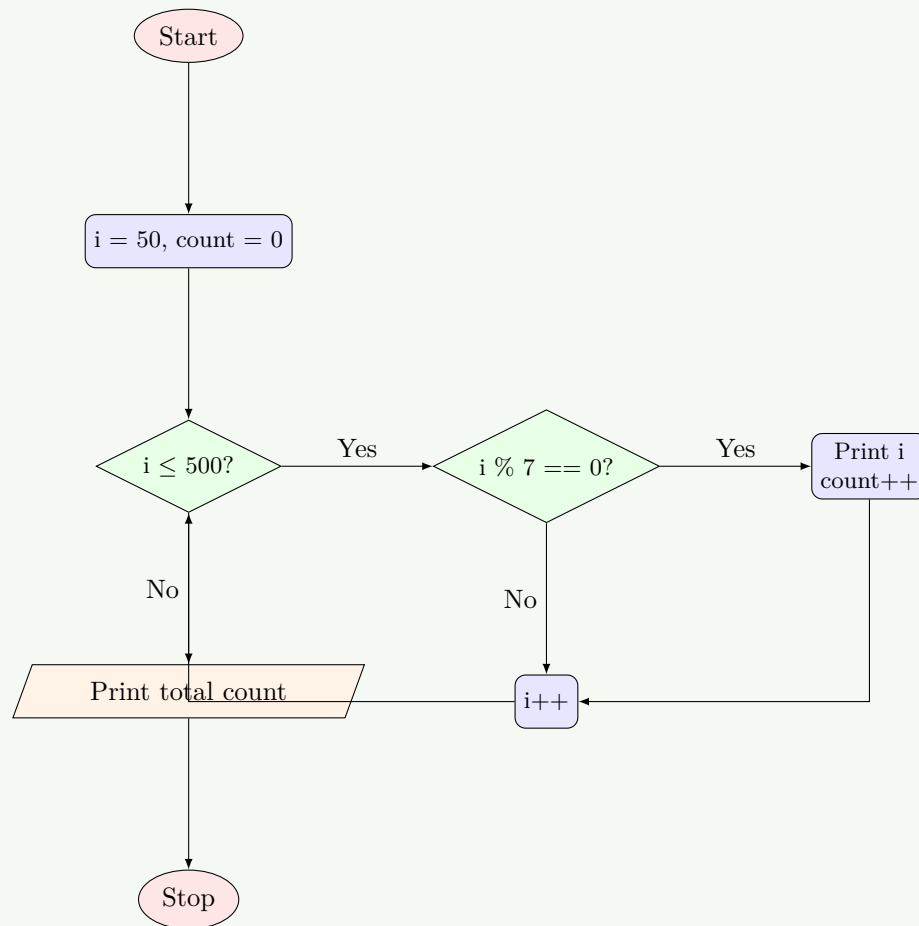
Answer:

```

1  #include <stdio.h>
2
3  int main() {
4      int i, count = 0;
5
6      printf("Numbers divisible by 7 between 50 and 500:\n");
7
8      // Find and print numbers divisible by 7
9      for(i = 50; i <= 500; i++) {
10         if(i % 7 == 0) {
11             printf("%d ", i);
12             count++;
13
14             // Print 10 numbers per line for better readability
15             if(count % 10 == 0)
16                 printf("\n");
17         }
18     }
19
20     printf("\nTotal count: %d\n", count);
21
22     return 0;
23 }

```

Algorithm Visualization:



- **Modulo operator:** $i \% 7 == 0$ checks divisibility
- **Formatting output:** Line breaks for readability
- **Counter variable:** Tracks how many numbers found

Mnemonic

"DIVide by SEVEN, ZERO remainder wins"

Question 5 [b marks]

4 Write a program which reads an integer from keyboard and prints whether given number is odd or even.

Solution

Answer:

```

1  #include <stdio.h>
2
3  int main() {
4      int number;
5
6      // Input an integer
7      printf("Enter an integer: ");
  
```

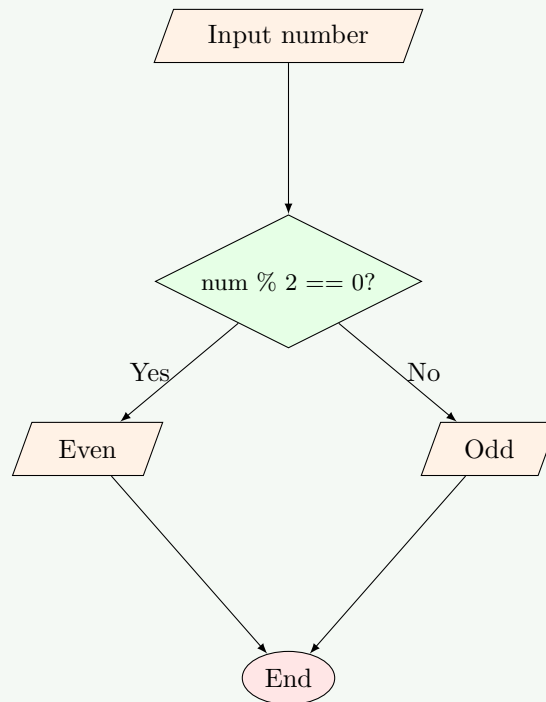


```

8   scanf("%d", &number);
9
10  // Check if the number is even or odd
11  if(number % 2 == 0) {
12      printf("%d is an even number.\n", number);
13  } else {
14      printf("%d is an odd number.\n", number);
15  }
16
17  return 0;
18  }

```

Decision Logic:



- **Modulo test:** Even numbers have remainder 0 when divided by 2
- **Binary representation:** Last bit is 0 for even, 1 for odd

Mnemonic

"EVEN with ZERO end, ODD with ONE bend"

Question 5 [c marks]

7 Define structure? Explain how it differs from array? Develop a structure named book to save following information about books. Book title, Name of author, Price and Number of pages.

Solution

Answer: A structure is a user-defined data type that allows grouping of variables of different data types under a single name.

Difference between Structure and Array:

Feature	Structure	Array
Data type	Can store different data types	Stores elements of same data type
Access	Members accessed using dot (.) operator	Elements accessed using index []
Memory allocation	Memory may not be contiguous	Memory is always contiguous
Size	Size can vary for each member	Size is same for all elements
Declaration	Uses struct keyword	Uses square brackets []
Purpose	Organizes related heterogeneous data	Organizes homogeneous data

Book Structure Program:

```

1  #include <stdio.h>
2  #include <string.h>
3
4  // Define the structure
5  struct Book {
6      char title[100];
7      char author[50];
8      float price;
9      int pages;
10 };
11
12 int main() {
13     // Declare a variable of type struct Book
14     struct Book myBook;
15
16     // Assign values to the structure members
17     strcpy(myBook.title, "C Programming");
18     strcpy(myBook.author, "Dennis Ritchie");
19     myBook.price = 350.50;
20     myBook.pages = 285;
21
22     // Display book information
23     printf("Book Details:\n");
24     printf("Title: %s\n", myBook.title);
25     printf("Author: %s\n", myBook.author);
26     printf("Price: %.2f\n", myBook.price);
27     printf("Pages: %d\n", myBook.pages);
28
29     return 0;
30 }

```

Structure Visualization:

struct Book myBook

title: "C Programming"
author: "Dennis Ritchie"
price: 350.50
pages: 285

- **Structure definition:** Creates template for data
- **Member access:** Use dot operator (structure.member)
- **String handling:** Uses string functions for character arrays

Mnemonic

"STRUCTURE groups DIFFERENT, ARRAY repeats SAME"

OR

Question 5 [a marks]

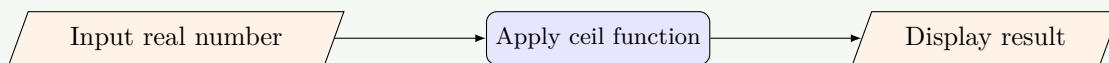
3 Write a program which reads a real number from keyboard and prints a smallest integer greater than it.

Solution

Answer:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     float number;
6     int result;
7
8     // Input a real number
9     printf("Enter a real number: ");
10    scanf("%f", &number);
11
12    // Find smallest integer greater than the input
13    result = ceil(number);
14
15    // Display the result
16    printf("Smallest integer greater than %.2f is %d\n", number, result);
17
18    return 0;
19 }
```

Function Behavior:



- **Math function:** `ceil()` rounds up to next integer
- **Result type:** Returns smallest integer greater than input
- **Handling edge cases:** Works with negative numbers

Mnemonic

"CEILING function, UP we go, NEXT integer we show"

OR

Question 5 [b marks]

4 Write a program which reads character from keyboard and prints its ASCII value.

Solution

Answer:

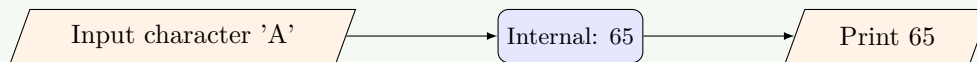
```
1 #include <stdio.h>
2
3 int main() {
4     char ch;
5
6     // Input a character
7     printf("Enter a character: ");
8     scanf("%c", &ch);
```

```

9
10 // Display ASCII value of the character
11 printf("ASCII value of '%c' is %d\n", ch, ch);
12
13 return 0;
14 }

```

Program Visualization:



- **Character storage:** Characters stored as integers in memory
- **Type conversion:** Automatic conversion from char to int
- **Extended ASCII:** Values from 0 to 255 for 8-bit characters

Mnemonic

"CHARS have NUMBERS underneath, PRINT shows BOTH sides"

OR

Question 5 [c marks]

7 Define function? Explain its advantage. Write function to calculate the square of a given integer number.

Solution

Answer: A function is a self-contained block of code designed to perform a specific task. It takes input, processes it, and returns an output.

Advantages of Functions:

Advantage	Description
Code reusability	Write once, use many times
Modularity	Break complex problems into manageable parts
Maintainability	Easier to debug and modify isolated code
Abstraction	Hide implementation details
Readability	Makes code more organized and understandable
Scope control	Variables local to functions reduce naming conflicts

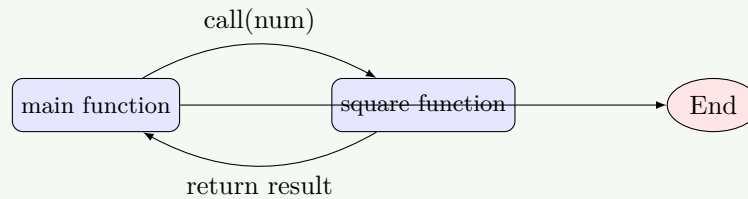
Program with Square Function:

```

1 #include <stdio.h>
2
3 // Function to calculate square of an integer
4 int square(int num) {
5     return num * num;
6 }
7
8 int main() {
9     int number, result;
10
11     // Input an integer
12     printf("Enter an integer: ");
13     scanf("%d", &number);
14
15     // Call the square function

```

```
16     result = square(number);  
17  
18     // Display the result  
19     printf("Square of %d is %d\n", number, result);  
20  
21     return 0;  
22 }
```

Function Flow:

- **Function prototype:** Declares function signature
- **Parameters:** Input values passed to function
- **Return value:** Output or result from function

Mnemonic

"Functions ENCAPSULATE tasks, take INPUTS, give OUTPUTS"