Internal assessment Test - 1

AMC ENGINEERING COLLEGE (AUTONOMOUS)
NAAC A+ | NBA Accredited
Affiliated to VTU & AICTE

USN | | | | | | | | | | |

Department of Computer Science & Engineering (AIML)

Internal Assessment – 1

1st Semester B. E – November 2025

Course Name: **Programming in C**	Max Marks: 50
Course Code: **1BEIT105**	Time: 90 mins

Note: Answer three full questions selecting one from each module

Q. No.	Question	Marks	COs	Level
1. a	Discuss the structure of "C" program with an example.	06	CO1	L2
b	What are variables? Write the rules to declare variables.	07	CO2	L2
c	What are escape sequence? Mention the escape sequences of "C" language with their meaning.	07	CO2	L3
	OR			
2. a	Explain any two output devices in details.	06	CO1	L2
b	With an example, explain Input – Output statements in "C"	07	CO2	L2
c	Apply the concept of the four C scopes in C programming to develop a small program that demonstrates how each scope works.	07	CO2	L3
3. a	Define Recursion. Mention the properties of Recursion function.	06	CO1	L2
b	What is the purpose of format specifiers in C? Give examples of commonly used format specifiers.	07	CO2	L2
c	Write a C program using a **switch** statement to perform a simple calculator operation (addition, subtraction, multiplication, division) based on the user's choice.	07	CO2	L3
	OR			
4. a	Differentiate between formatted and unformatted I/O functions.	06	CO1	L2
b	What is the purpose of the "if" and "while" statements? Explain with example.	07	CO2	L2
c	Write a C program to find the sum of first 10 natural numbers using while loop.	07	CO2	L3
5. a	Write a "C" program to sort the array of elements in ascending order.	05	CO3	L2
b	How to pass Single dimensional arrays? Explain with example.	05	CO3	L3
	OR			
6. a	Write a program to find the transpose of matrix.	05	CO3	L2
b	How 2 Dimensional arrays are declared and initialized?	05	CO3	L3

Prepared by Course Teacher/ Course Coordinator	Scrutinized by Module Coordinator / Program Coordinator/PAC member	Approved by Academic Advisor/PAC
Signature:	Signature:	Signature:
Name: **Prof. Parthasarathy P V**	Name: **Dr. Mareeswari V**	Name: **Dr. Yuvaraju B N**

Internal assessment Test - 1

USN										

AMC ENGINEERING COLLEGE (AUTONOMOUS)
NAAC A+ | NBA Accredited
Affiliated to VTU & AICTE

Department of Computer Science & Engineering (AIML)

Internal Assessment – 1

1st Semester B. E – November 2025

Course Name: **Programming in C**	Max Marks: 50
Course Code: **1BEIT105**	Time: 90 mins

Note: Answer three full questions selecting one from each module

Q. No.	Question	Marks	COs	Level
1. a	Compare Compilers with Interpreters	06	CO1	L2
b	Explain the following: (i) Compiler (ii) Linking (iii) Memory map.	07	CO2	L2
c	List out the scopes that determine the visibility of an identifier with meaning.	07	CO2	L3
	OR			
2. a	Explain arithmetic operators, Increment and Decrement operators with example.	06	CO1	L2
b	What is Expression? Write with suitable example.	07	CO2	L2
c	Apply relational and logical operators in a C program to compare values and make decisions. Demonstrate the use of each operator with suitable examples and output.	07	CO2	L3
3. a	Write a program to check whether the given number is palindrome or not.	06	CO2	L2
b	What are all the console input and output functions used in “C”.	07	CO1	L2
c	Apply the ternary operator in a C program to make a decision between two values and display the result with a suitable example.	07	CO2	L3
	OR			
4. a	Explain block statements with example.	06	CO1	L2
b	What is the purpose of the “while” and “do..while” statements? Explain with example.	07	CO1	L2
c	Write a C program to find the roots of the quadratic equation.	07	CO3	L3
5. a	Write a “C” program to sort the array of elements in descending order.	05	CO3	L2
b	Apply the concept of passing two-dimensional arrays to a function in C and demonstrate it with a suitable program and output.	05	CO3	L3
	OR			
6. a	Write a C program to sort given set of 10 integers using arrays.	05	CO3	L2
b	Apply the concept of one-dimensional arrays to store and display strings in a C program. Demonstrate using a suitable example.	05	CO3	L3

Prepared by Course Teacher/ Course Coordinator	Scrutinized by Module Coordinator / Program Coordinator/PAC member	Approved by Academic Advisor/PAC
Signature:	Signature:	Signature:
Name: **Prof. Parthasarathy P V**	Name: **Dr. Mareeswari V**	Name: **Dr. Yuvaraju B N**

Internal assessment Test - 1

AMC ENGINEERING COLLEGE (AUTONOMOUS)
NAAC A+ | NBA Accredited
Affiliated to VTU & AICTE

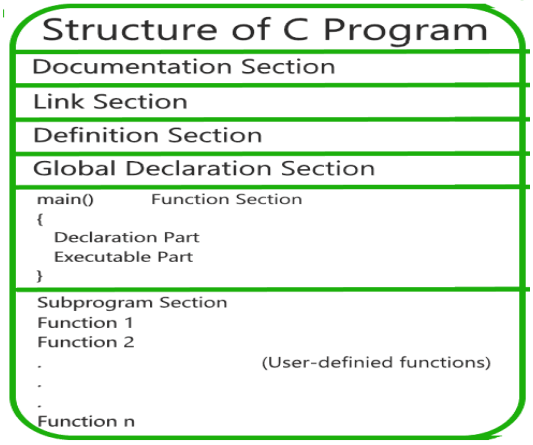
Department of Computer Science & Engineering
Internal Assessment – 1
1st Semester B. E – November 2025

Course Name: **Programming in C**	Max Marks: 50
Course Code: **1BEIT105**	Time: 90 mins

QUESTIONS & ANSWERS

Q. No.	Question & Answer
1. a	***Discuss the structure of "C" program with an example.***

The Structure of a "C" Program



Documentation Section: This section contains comments that describe the program — its purpose, author, date, version, etc.
Example:

```
/*
  Program: Calculate Area of a Circle
  Author: Prof. Parthasarathy P V
  Date: 13/11/2025
  Description: This program calculates the area of a circle using radius.
*/
```

Link Section: Used to include header files that contain declarations for built-in functions (like printf(), scanf(), etc.).
Example:

```
#include <stdio.h>     // Standard Input/Output functions
#include <math.h>      // Mathematical functions
```

Definition Section: Defines constants and macros that are used throughout the program.
Example:

```
#define PI 3.14159
#define MAX 100
```

Global Declaration Section: Declares variables and functions that are accessible from any part of the program.
Example:

```
int total;         // Global variable
float sum(float, float);   // Function declaration (prototype)
```

Main Function Section: This is the **entry point** of every C program. Execution begins from here.

```
int main()
```

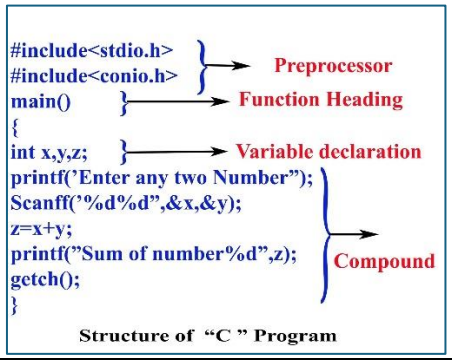
```
{
   // Declaration part
   float radius, area;

   // Executable part
   printf("Enter the radius: ");
   scanf("%f", &radius);

   area = 3.14159 * radius * radius;
   printf("Area of Circle = %.2f", area);

   return 0;
}
```

Subprogram Section (User-Defined Functions): Contains additional functions defined by the user to modularize code.

Syntax Example:

```
float sum(float x, float y)
{
   return (x + y);
}
```

Section	Purpose	Example
Documentation	Program description	/* ... */
Link	Include libraries	#include <stdio.h>
Definition	Define constants/macros	#define PI 3.14
Global Declaration	Declare global variables/functions	int sum(int,int);
Main Function	Start of execution	int main() { ... }
Subprograms	User-defined functions	float area(float r) { ... }



Structure of "C" Program

1.b

What are variables? Write the rules to declare variables.

Variables

- A variable is a named location in memory that is used to hold a value that can be modified by the program.
- All variables must be declared before they can be used.
- The general form: ***type variable_list;***
  Here, ***type*** must be a valid data type plus any modifiers
  variable_list may consist of one or more identifier names separated by commas.
- Here are some declarations:
  int i, j, l;
  short int si;

unsigned int ui;
double balance, profit, loss;

Where Variables Are Declared

- Variables can be declared in ***three*** places:
  1. ***Inside*** functions - ***Local variables***
  2. ***In the definition*** of function parameters - ***Formal parameters***
  3. ***Outside*** of all functions ***- Global variables***

(1) Local variables

- Variables that are declared inside a function are called local variables (automatic variables).
- Local variables can be used only by statements that are inside the block in which the variables are declared (not known outside their own code block).

Example

void func1(void) ***{*** ***int x;*** ***x = 10;*** ***}*** ***void func2(void)*** ***{*** ***int x;*** ***x = -199;*** ***}***	The integer variable x is declared twice, once in func1() and once in func2( ). The x in func1( ) has no bearing on or relationship to the x in func2( ). As explained, this is because each x is known only to the code within the block in which it is declared.

(2) Formal parameters

- ***If a function is to use arguments***, it ***must declare variables*** that will accept the values of the arguments. ***These variables are called the formal parameters of the function***. They ***behave like any other local variables inside the function***. As shown in the following program fragment, their declarations occur after the function name and inside parentheses.

```
/* Return 1 if c is part of string s; 0 otherwise */
int is_in(char *s, char c)
{
  while(*s)
    if(*s==c) return 1;
    else s++;
  return 0;
}
```

(3) Global variables

- ***Global variables are known throughout the program and may be used by any piece of code***. Also, they will ***hold their value throughout the program's execution***.
- You ***create global variables by declaring them outside of any function***. Any expression may access them, regardless of what block of code that expression is in.

```
#include <stdio.h>
int count;  /* count is global  */

void func1(void);
void func2(void);

int main(void)
{
  count = 100;
  func1();

  return 0;
}

void func1(void)
{
  int temp;

  temp = count;
  func2();
  printf("count is %
d", count); /* will print 100 */
}
```


1.c

What are escape sequence? Mention the escape sequences of "C" language with their meaning.

Backslash Character Constants (carriage return)

- ***"C"*** includes the special backslash character constants, so that you may easily enter these special characters as constants. These are also referred to as ***escape sequences***. You should ***use the backslash codes instead of their ASCII equivalents to help ensure portability***.
- For example, the following program ***outputs a new line and a tab and then prints the string*** **"This is a test".**

```
#include <stdio.h>
int main(void)
{
  printf("\n\tThis is a test.");
  return 0;
}
```

Table *Backslash Codes*

Code	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert
\?	Question mark
\N	Octal constant (where N is an octal constant)
\xN	Hexadecimal constant (where N is a hexadecimal constant)

2. a

Explain any two output devices in details.

1. Monitor (Visual Display Unit)

Definition:

A **monitor** is the most common **output device** that displays text, images, videos, and graphical data from the computer. It is also known as a **Visual Display Unit (VDU)**.

Working Principle:

- The computer's **video card (graphics card)** sends electronic signals to the monitor.
- These signals are converted into images or text that appear on the screen.

Types of Monitors:

1. **CRT (Cathode Ray Tube) Monitor:**
    - Older type of monitor (bulky and heavy).
    - Uses electron beams to excite phosphor dots on the screen to form images.
    - Example: Used in old desktop computers and televisions.
2. **LCD (Liquid Crystal Display) Monitor:**
    - Lightweight, flat, and energy-efficient.
    - Uses liquid crystals and fluorescent light to produce images.
    - Common in laptops and modern desktops.
3. **LED (Light Emitting Diode) Monitor:**
    - Improved version of LCD.
    - Uses LEDs for backlighting, giving better contrast and brightness.
    - More eco-friendly and has better color accuracy.

Advantages:

- Provides instant visual feedback.
- Allows viewing of both text and graphics.
- Energy-efficient and space-saving (modern models).

2. Printer

Definition:

A **printer** is an **output device** that produces a **hard copy** (permanent record) of data from the computer on paper.

Working Principle:

- The computer sends print commands as digital data.
- The printer converts the digital data into printed text, symbols, or graphics on paper.

Types of Printers:

A. Impact Printers

These printers physically strike the paper to print characters.

1. **Dot Matrix Printer:**
    - Prints characters as a pattern of dots.
    - Uses a ribbon and print head.
    - Noisy but can print carbon copies.
    - Used in banks and billing counters.
2. **Line Printer:**
    - Prints one line at a time.
    - Very fast and used for large volume printing (mainframes).

B. Non-Impact Printers

These printers do **not physically strike** the paper.

1. **Inkjet Printer:**
    - Sprays tiny drops of ink onto paper.
    - Produces high-quality colored output.
    - Commonly used in homes and offices.
2. **Laser Printer:**
    - Uses laser beams and toner powder.
    - Produces high-speed and high-quality prints.
    - Suitable for business and professional use.

Advantages:

- Produces permanent copies of data.
- High-quality text and image output.
- Color printing possible (in inkjet/laser printers).

2.b

With an example, explain Input – Output statements in "C"

Input and Output Statements in C

- C language uses special library functions to perform input (reading data) and output (displaying data) operations.
  These functions are defined in the header file <stdio.h>, which stands for Standard Input and Output.

Input and Output Statements in C Language

C language provides standard input and output functions to read and display data. These functions are defined in the header file <stdio.h> (Standard Input Output Header).

1. Output Statement – printf()

The printf() function is used to display data or messages on the screen.

Syntax:

printf("format string", arguments);

Description: The format string specifies how the output will appear, and the arguments represent the variables or values to display.

Example of printf():

```
#include <stdio.h>
int main()
{
   int age = 20;
   float marks = 85.5;
   char grade = 'A';

   printf("Age: %d\n", age);
   printf("Marks: %.2f\n", marks);
   printf("Grade: %c\n", grade);

   return 0;
}
```

Output:

Age: 20

Marks: 85.50

Grade: A

2. Input Statement – scanf()

The scanf() function is used to take input from the user via the keyboard.

Syntax:

scanf("format string", &variable);

Description: The format string specifies the type of data expected. The & symbol is used before variable names (except for strings) to store the input value in memory.

Example of scanf():

```
#include <stdio.h>
int main()
{
   int age;
   float marks;
   char name[20];

   printf("Enter your name: ");
   scanf("%s", name);

   printf("Enter your age: ");
   scanf("%d", &age);
```

```
    printf("Enter your marks: ");
    scanf("%f", &marks);

    printf("\n--- Student Details ---\n");
    printf("Name: %s\nAge: %d\nMarks: %.2f\n", name, age, marks);

    return 0;
}
```

Sample Output:
Enter your name: John
Enter your age: 18
Enter your marks: 92.5

--- Student Details ---
Name: John
Age: 18
Marks: 92.50

Summary Table

Function	Purpose	Syntax Example	Used For
printf()	Displays output on screen	printf("Sum = %d", sum);	Output
scanf()	Reads input from user	scanf("%d", &num);	Input

2.c

Apply the concept of the four C scopes in C programming to develop a small program that demonstrates how each scope works.

Concept of the four C scopes in C programming

- Standard ***"C"*** defines four scopes that determine the visibility of an identifier.

Scope	Meaning
1. File scope	• Starts at the beginning of the file and ends with the end of the file. • It refers only to those identifiers that are declared outside of all functions. • File scope identifiers are visible throughout the entire file. • Variables that have file scope are global.
2. Block scope	• Begins with the opening ***"{"*** of a block and ends with its associated closing ***"}"***. • Block scope also extends to function parameters in a function definition. That is, function parameters are included in a function's block scope. • Variables with block scope are local to their block.
3. Function prototype scope	• Identifiers declared in a function prototype; visible within the prototype.
4. Function scope	• Begins with the opening ***"{"*** of a function and ends with its closing ***"}"***. • ***Function scope applies only to labels***.

- A label is used as the target of a ***"goto"*** statement, and that label must be within the same function as the ***"goto"***.

3. a

Define Recursion. Mention the properties of Recursion function.

Recursion

- Recursion is a process in which a function calls itself directly or indirectly to solve a problem.
  It breaks a large problem into smaller sub-problems of the same type.
- In C, a recursive function must have:
  - Base case – the condition under which the function stops calling itself.
  - Recursive case – the part where the function calls itself again.

General Form of a Recursive Function

```
return_type function_name(parameters)
{
   if (base_condition)
      return value;
   else
      return function_name(modified_parameters);
}
```

Simple Example: Factorial of a Number

```
#include <stdio.h>

int factorial(int n)
{
   if (n == 0)        // Base case
      return 1;
   else
      return n * factorial(n - 1);  // Recursive call
}

int main()
{
   int num;
   printf("Enter a number: ");
   scanf("%d", &num);

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

   return 0;
}
```

Explanation:

Let's say num = 4

Function Call	Computation	Result Returned
factorial(4)	4 × factorial(3)	waits for factorial(3)
factorial(3)	3 × factorial(2)	waits for factorial(2)
factorial(2)	2 × factorial(1)	waits for factorial(1)
factorial(1)	1 × factorial(0)	waits for factorial(0)
factorial(0)	returns 1 (base case)	returns 1
Backtrack	factorial(1)=1×1=1 → factorial(2)=2×1=2 → factorial(3)=3×2=6 → factorial(4)=4×6=24	✅ Final Result = 24

Key Points:

- Each recursive call creates a new copy of the function on the call stack.
- Base condition prevents infinite recursion.
- Recursive functions are elegant but may use more memory than iterative loops.

3.b

What is the purpose of format specifiers in C? Give examples of commonly used format specifiers.

Format specifiers in C

- The input format specifiers are preceded by a "%" sign and tell scanf( ) what type of data is to be read next. These codes are listed in Table. The format specifiers are matched, in order from left to right, with the arguments in the argument list.

Table. *scanf( ) Format Specifiers*

Code	Meaning
%a	Reads a floating-point value (C99 only).
%c	Reads a single character.
%d	Reads a decimal integer.
%i	Reads an integer in either decimal, octal, or hexadecimal format.
%e	Reads a floating-point number.
%f	Reads a floating-point number.
%g	Reads a floating-point number.
%o	Reads an octal number.
%s	Reads a string.
%x	Reads a hexadecimal number.
%p	Reads a pointer.
%n	Receives an integer value equal to the number of characters read so far.
%u	Reads an unsigned decimal integer.
%[ ]	Scans for a set of characters.
%%	Reads a percent sign.

3. c

Write a C program using a switch statement to perform a simple calculator operation (addition, subtraction, multiplication, division) based on the user's choice.

Program using "switch" statement

```
#include <stdio.h>

int main() {
    char operator;
    double num1, num2;

    printf("Enter an operator (+, -, *, /): ");
    scanf(" %c", &operator); // Note the space before %c to consume leftover newline
character
```

```
    printf("Enter two numbers: ");
    scanf("%lf %lf", &num1, &num2);

    switch (operator) {
        case '+':
            printf("%.2lf + %.2lf = %.2lf\n", num1, num2, num1 + num2);
            break;
        case '-':
            printf("%.2lf - %.2lf = %.2lf\n", num1, num2, num1 - num2);
            break;
        case '*':
            printf("%.2lf * %.2lf = %.2lf\n", num1, num2, num1 * num2);
            break;
        case '/':
            if (num2 != 0) {
                printf("%.2lf / %.2lf = %.2lf\n", num1, num2, num1 / num2);
            } else {
                printf("Error: Division by zero is not allowed.\n");
            }
            break;
        default:
            printf("Error: Invalid operator.\n");
    }

    return 0;
}
```

4. a

Differentiate between formatted and unformatted I/O functions.

Program: Sort an Array in Ascending Order

I/O Functions in C Language

Input and Output (I/O) functions in C are used to read data from the user (input) and display data to the screen (output).
They are defined in the <stdio.h> (Standard Input/Output) header file.
These functions are mainly divided into two types:

1. *Formatted I/O Functions*
2. *Unformatted I/O Functions*

Formatted I/O Functions

Definition:
Formatted I/O functions allow the user to control the input and output format of data using format specifiers (like %d, %f, %s, etc.).
Common Functions:

- *printf() → Displays formatted output*
- *scanf() → Reads formatted input*

```
#include <stdio.h>
int main()
{
   int age;
   float marks;

   printf("Enter age and marks: ");
```

```
    scanf("%d %f", &age, &marks);

    printf("Age = %d, Marks = %.2f\n", age, marks);

    return 0;
}
```

Unformatted I/O Functions

Unformatted I/O functions are used for character-by-character or string-based input and output without using format specifiers.
They are simpler but less flexible.
Common Functions:

Function	Purpose
getchar()	Reads a single character from the keyboard
putchar()	Displays a single character on the screen
gets()	Reads a string (until newline)
puts()	Displays a string on the screen

Example:

```
#include <stdio.h>
int main()
{
    char name[20];

    printf("Enter your name: ");
    gets(name);          // Read string (unformatted)

    printf("Hello ");
    puts(name);          // Display string (unformatted)

    return 0;
}
```

4.b

What is the purpose of the "if" and "while" statements? Explain with example.

Program: Sort an Array in Ascending Order

"if" statement

- The general form of the if statement is

  if (expression)
  statements;
  else
  statements;

  where a ***statement*** may ***consist*** of a ***single*** statement, a ***block of statements***, or ***nothing*** (in the case of empty statements). The ***"else"*** clause is optional.
- If expression ***evaluates to true*** (anything other than 0),
  - the ***statement or block that forms*** the target of ***"if"*** is executed;
  - Otherwise, the statement or block that is the target of ***"else"*** will be executed, if it exists.
- ***Remember***, only the code associated with ***"if"*** or the code associated with ***"else"*** executes, **never both**.

- The conditional statement controlling ***"if"***must ***produce a scalarresult***.
- A ***scalar*** is either
  - An integer
  - Character
  - Pointer, or
  - Floating-point(rare to use - because this slows execution time considerably - takes several instructions to perform).
  - Bool(C99)
  - ***Use few instructions*** to ***perform an integer*** or ***character operation***.
- The following program contains an example of***"if"***.

```
/* Magic number program */
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
  int magic; /* magic number */
  int guess; /* user's guess */

  magic = rand(); /* generate the magic number */

  printf("Guess the magic number: ");
  scanf("%d", &guess);
  if(guess == magic)
     printf("** Right **");
  else printf("Wrong");
  return 0;
}
```

Nested "if"s

- A nested ***"if"*** is an ***"if"*** that is the target of another ***"if"*** or ***"else"***. Nested ***"if"***s are very common in programming. In a nested ***"if"***, an ***"else"***statement always refers to the nearest ***"if"*** statement that is within the same block as the ***"else"***and that is not already associated with an ***"else"***.

```
if(i)
{
  if(j) dosomething1();
  if(k) dosomething2(); /* this if */
  else  dosomething3(); /* associated with this else*/
}
else dosomething4(); /* associated with if(i) */
```

- For example:

The "while" Loop

- General form

  while(condition)
  statement;

  condition may be any expression, and ***true*** is any ***nonzero*** value.
  The ***loop iterates*** while the ***condition is true***.
  When the ***condition becomes false***, ***program control passes to the line of code immediately following the loop***.
- The following example shows a keyboard input that simply loops until the user types ***"A":***

```c
#include <stdio.h>

int main() {
    char ch = '\0';
    printf("Enter a character: ");
    while(ch != 'A') ch = getchar();
    printf("After you entered 'A' While Loop
        terminated ");
    return 0;
}
```

Output

```
Enter a character: 2
9
a
A
After you entered 'A' While Loop terminated
```


- First, **"ch"** is initialized to null. The while loop then checks to see if **"ch"** is not equal to **"A"**. Because **"ch"** was initialized to null, the test is **true** and the **loop begins**. Each time you press a key, the condition is tested again. Once you enter an **"A"**, the condition becomes false because **"ch"** equals **"A"**, and the loop terminates.
- Like **"for"** loops, **"while"** loops check the test condition at the top of the loop, which means that the body of the loop will not execute if the condition is false to begin with.

```c
/* Add spaces to the end of a string. */
void pad(char *s, int length)
{
  int l;

  l = strlen(s); /* find out how long it is */

  while(l < length) {
    s[l] = ' '; /* insert a space */
    l++;
  }
  s[l]= '\0'; /* strings need terminated in a null*/
}
```

- In cases in which any one of several separate conditions can terminate a while loop, often a single loop-control variable forms the conditional expression. The value of this variable is set at various points throughout the loop. In this example

```c
void func1(void)
{
  int working;

  working = 1; /* i.e., true*/

  while (working) {
    working = process1();
    if (working)
       working = process2();
    if (working)
       working = process3();
  }
}
```

- Any of the three routines may return false and cause the loop to exit.
- There need not be any statements in the body of the while loop. For example,

  while((ch=getchar()) != 'A');

  will simply loop until the user types **"A".** If you feel uncomfortable putting the assignment inside the **"while"** conditional expression, remember that the equal sign is just an operator that evaluates to the value of the right-hand operand.

4.c	***Write a C program to find the sum of first 10 natural numbers using while loop.*** **Program: Sum of First 10 Natural Numbers Using While Loop** ***#include <stdio.h>*** ***int main()*** ***{*** ***int i = 1, sum = 0;*** ***// Using while loop to calculate sum*** ***while (i <= 10)*** ***{*** ***sum = sum + i; // Add current number to sum*** ***i++; // Increment the counter*** ***}*** ***printf("Sum of first 10 natural numbers = %d\n", sum);*** ***return 0;*** ***}*** ***Explanation:*** • ***Initialization:*** ***i = 1 (the first natural number).*** ***sum = 0 (to store the total).*** • ***Condition:*** ***The loop runs while i <= 10 — that is, until i becomes 11.*** • ***Body of Loop:*** ***Each iteration adds i to sum.*** • ***Increment:*** ***i++ increases i by 1 on every pass.*** • ***After loop ends:*** ***The variable sum holds the total of the first 10 natural numbers.***
5. a	***Write a "C" program to sort the array of elements in ascending order.*** **Program: Sort an Array in Ascending Order** ***#include <stdio.h>*** ***int main()*** ***{*** ***int a[100], n, i, j, temp;*** ***// Step 1: Input number of elements***

```
    printf("Enter number of elements: ");
    scanf("%d", &n);
    // Step 2: Input array elements
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++)
    {
      scanf("%d", &a[i]);
    }
    // Step 3: Sorting logic (Bubble Sort)
    for (i = 0; i < n - 1; i++)
    {
      for (j = i + 1; j < n; j++)
      {
        if (a[i] > a[j])  // Swap if elements are not in order
        {
          temp = a[i];
          a[i] = a[j];
          a[j] = temp;
        }
      }
    }
    // Step 4: Display sorted array
    printf("\nArray in Ascending Order:\n");
    for (i = 0; i < n; i++)
    {
      printf("%d ", a[i]);
    }
    return 0;
  }
```

Explanation of the Program

1. Input Phase:
   - The user enters how many elements are in the array (n).
   - Then enters n numbers.
2. Sorting Phase:
   - We compare each element with the others.
   - If a larger element appears before a smaller one, they are swapped.
   - This process continues until all elements are arranged in ascending order.
3. Output Phase:
   - The sorted elements are printed.

5.b

How to pass Single dimensional arrays? Explain with example.

Single-Dimension Arrays

General Form	Example
type var_name[size];	double balance[100];

- Like other variables, arrays must be explicitly declared so that the compiler can allocate space for them in memory. Here, "***type***" declares the ***base type of the array***, which is the type of each element in the array, and size defines how many elements the array will hold.
- For example, this code will compile without error, but it is incorrect because the for loop will cause the array count to be overrun.

```
int count[10], i;
/* this causes count to be overrun */
for(i=0; i<100; i++) count[i] = i;
```

- Single-dimension arrays are essentially lists that are stored in contiguous memory locations in index order. For example, Figure shows how array a appears in memory if it starts at memory location 1000 and is declared as shown here***: char a[7];***

Element	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
Address	1000	1001	1002	1003	1004	1005	1006

Passing Single-Dimension Arrays to Functions

- You can also specify the address of the first element of an array by using the & operator. For example, ***sample*** and ***&sample[0]*** both produce the same results. However, in professionally written C code, you will almost never see ***&sample[0].***
- In C, you cannot pass an entire array as an argument to a function. You can, however, pass a pointer to an array by specifying the array's name without an index.
- For example, the following program fragment passes the address of ***"i"*** to ***"func1( )":***

```
int main(void)
{
  int i[10];
  func1(i);
  /* . . . */
}
```

- If a function receives a pointer to a single-dimension array, you can declare its formal parameter in one of three ways: as a pointer, as a sized array, or as an unsized array.
- For example, to receive ***"i",*** a function called ***func1( )*** can be declared as

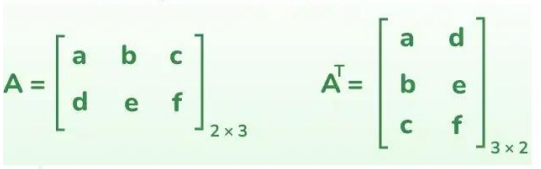
void func1(int *x) {	***void func1(int x[10]) {***	***void func1 (int x[]){***
/ pointer */*	*/* sized array */*	*/* unsized array */*
/*. . . */	*/*. . . */*	*/*. . . */*
}	***}***	***}***

- ***All three declaration methods produce similar results*** because each tells the compiler that an ***integer pointer*** is going to be ***received***.
-

6. a

Write a program to find the transpose of matrix.

Program to find the transpose of matrix




```
#include <stdio.h>

int main()
{
   int a[10][10], transpose[10][10];
   int row, col, i, j;

   // Step 1: Input rows and columns
   printf("Enter number of rows and columns: ");
   scanf("%d %d", &row, &col);

   // Step 2: Input matrix elements
   printf("Enter elements of the matrix:\n");
   for (i = 0; i < row; i++)
   {
      for (j = 0; j < col; j++)
      {
         scanf("%d", &a[i][j]);
      }
   }

   // Step 3: Find transpose
   for (i = 0; i < row; i++)
   {
      for (j = 0; j < col; j++)
      {
         transpose[j][i] = a[i][j];
      }
   }

   // Step 4: Display original matrix
   printf("\nOriginal Matrix:\n");
   for (i = 0; i < row; i++)
   {
      for (j = 0; j < col; j++)
      {
         printf("%d\t", a[i][j]);
      }
      printf("\n");
   }

   // Step 5: Display transpose matrix
   printf("\nTranspose of the Matrix:\n");
   for (i = 0; i < col; i++)
   {
      for (j = 0; j < row; j++)
      {
         printf("%d\t", transpose[i][j]);
```

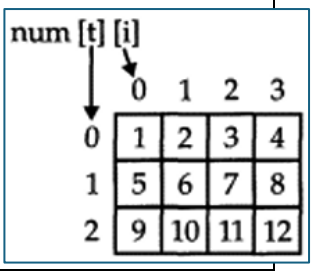
```
        }
        printf("\n");
    }

    return 0;
}
```

6.b

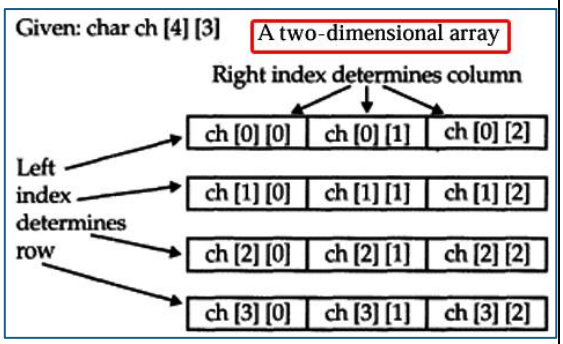
How 2 Dimensional arrays are declared and initialized?

Two-Dimensional Arrays

- C supports multidimensional arrays. The simplest form of the multidimensional array is the two dimensional array.
- To declare a two-dimensional integer array d of size 10,20, you would write int ***d[10][20];***
- Similarly, to ***access point 1,2 of array d***, you would use ***d[1] [2].***
- The following example loads a two-dimensional array with the numbers 1 through 12 and prints them row by row.

Program	Output
`#include <stdio.h>` `int main(void)` `{` `  int t, i, num[3][4];` `  for(t=0; t<3; ++t)` `    for(i=0; i<4; ++i)` `      num[t][i] = (t*4)+i+1;` `  /* now print them out */` `  for(t=0; t<3; ++t) {` `    for(i=0; i<4; ++i)` `      printf(''%3d ", num[t][i]);` `    printf("\n");` `  }` `  return 0;` `}`	In this example, ***num[0][0]*** value 1, ***num[0][1]*** the valu ***num[0][2]*** the value 3, and The value of ***num[2][3]*** will You can visualize the num shown here: 

- Two-dimensional arrays are stored in a row-column matrix, where
  - the ***left index*** indicates the ***row*** and
  - the ***right indicates*** the ***column***.



- This means that the rightmost index changes faster than the leftmost when accessing the elements in the array in the order in which they are actually stored in memory. See Figure for a graphic representation of a two-dimensional array in memory.
- In the case of a two-dimensional array, the following ***formula yields the number of bytes of memory needed to hold it:***

  bytes = size of 1st index × size of 2nd index × sizeof(base type)

- Therefore, assuming
  - integers = 4-byte
  - an integer array with dimensions [10][5] would have
  - bytes allocated = 10 × 5 × 4 = 200 bytes.
- When a ***two-dimensional array is used as an argument to a function, only a pointer to the first element is actually passed***. The parameter receiving a two-

	dimensional array must define at least the size of the rightmost dimension. The rightmost dimension is needed because the compiler needs to know the length of each row if it is to index the array correctly. • For example, a function that receives a two-dimensional integer array with dimensions [10][10] can be declared like this: ***void func1(int x[] [10]) {*** ***/* . . . */*** ***}*** • The compiler needs to know the size of the right dimension in order to correctly execute expressions such as ***x[2] [4]*** inside the function. ***If the length of a row is not known, the compiler cannot determine where the next row begins***.

Prepared by **Prof. Parthasarathy P V, Sudharani D, Spoorti R (I year "E")**