PSUC/PPS WORKSHOP

By

Journey of Learning Programming

- **Dive into the Code:** The secret to mastering programming lies in immersing yourself in the process. By actively writing code, you engage in hands-on learning, which is essential for understanding the intricacies of programming.**
-
- **Embrace the Errors:** When you compile your code, expect to encounter mistakes. These errors serve as invaluable learning opportunities, guiding you to refine and improve your code.
-
- **Refine and Improve:** Each time you compile and debug your code, you're not just correcting errors; you're also enhancing your programming knowledge and skills, steadily advancing your expertise.

- **Persistence Pays Off:** Writing code and addressing errors requires patience and resilience. However, this process builds a robust and determined mindset that is essential for long-term success in programming.

average programming experience

Basic Points to Remember for your Midsem

1. Always Write Something

- Even if you don't know the full answer, write something related. Partial answers may earn some marks.

2. Importance of Theory

- Theory questions can carry **5-6 marks** or more. Understanding the underlying concepts is essential, not just the code.

3. Focus on Logic Over Syntax

- When writing code, focus on the **core logic**. Even if the syntax isn't perfect, clear logic can fetch marks.

4. Practice on a Compiler

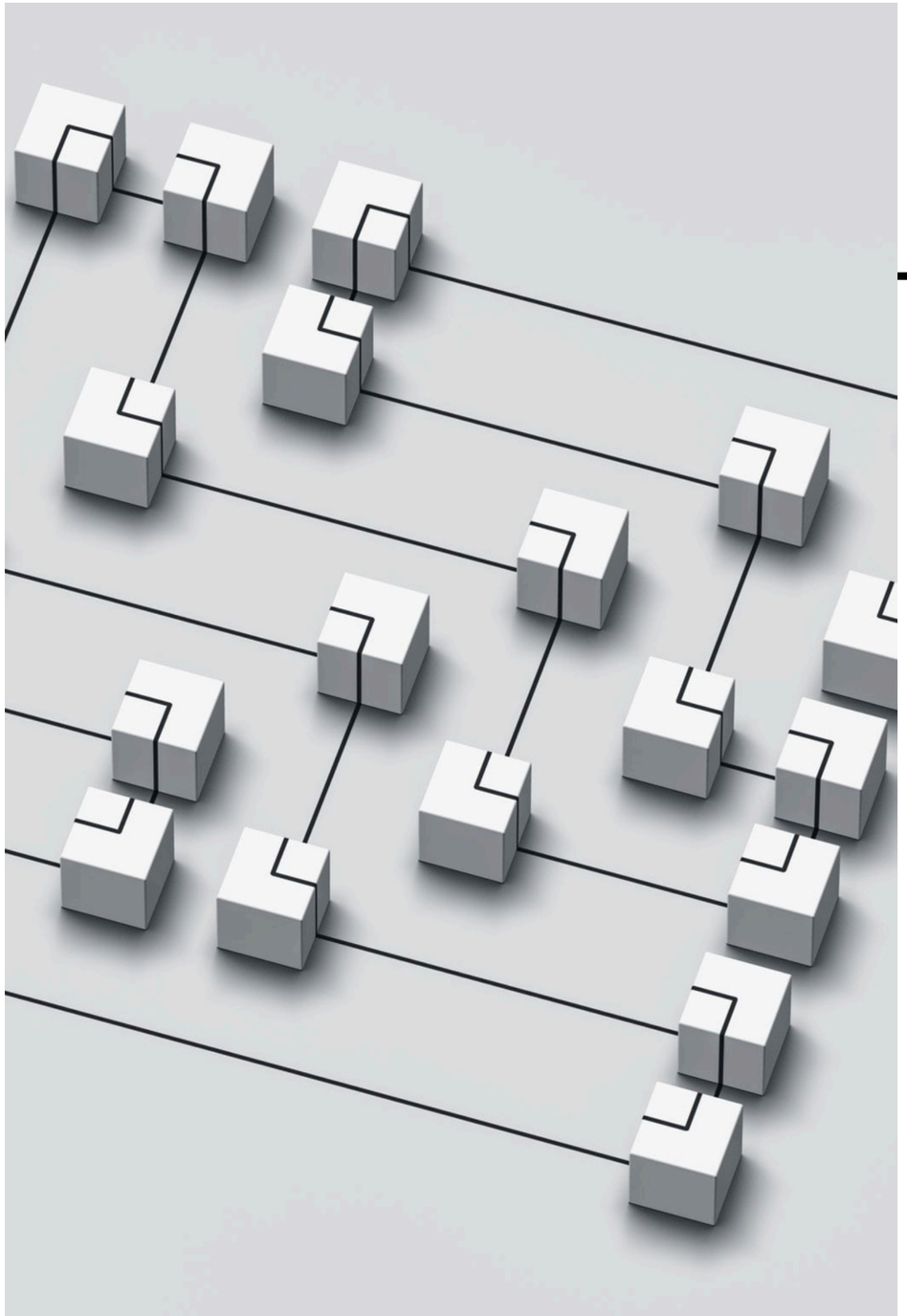
- Make sure to **practice coding** on an actual compiler to check for errors and improve your syntax.

5. Make Use of Provided Formula Booklet

- a formula booklet is provided during the exam, ensure you're familiar with it beforehand. It can help save time and avoid mistakes in recalling complex formulas, syntax and tables.

Algorithm

- An algorithm is a step-by-step procedure or set of rules designed for solving a specific problem or accomplishing a particular task.
- It serves as a blueprint for performing a task or solving a problem in a systematic and logical manner.



Flowchart

- A flowchart is a visual representation of a process or algorithm, using different shapes and arrows to illustrate the steps and their sequence
- It serves as a tool for designing, analyzing, and communicating the steps in a process

Steps to write an ALGORITHM

Name: Every Algorithm should have a name

Start/Stop:- Algorithm should begin with START and terminate with STOP

Steps:- Every Step must have a step number.

Clear Instructions:- All steps must be small , clear and simple .

Comments:- Every step can have a comment enclosed in [ ] to increase readability.

Properties of an Algorithm

- Finite
- Definite
- Input / Output
- Effective

Algorithm to find the factorial of a number

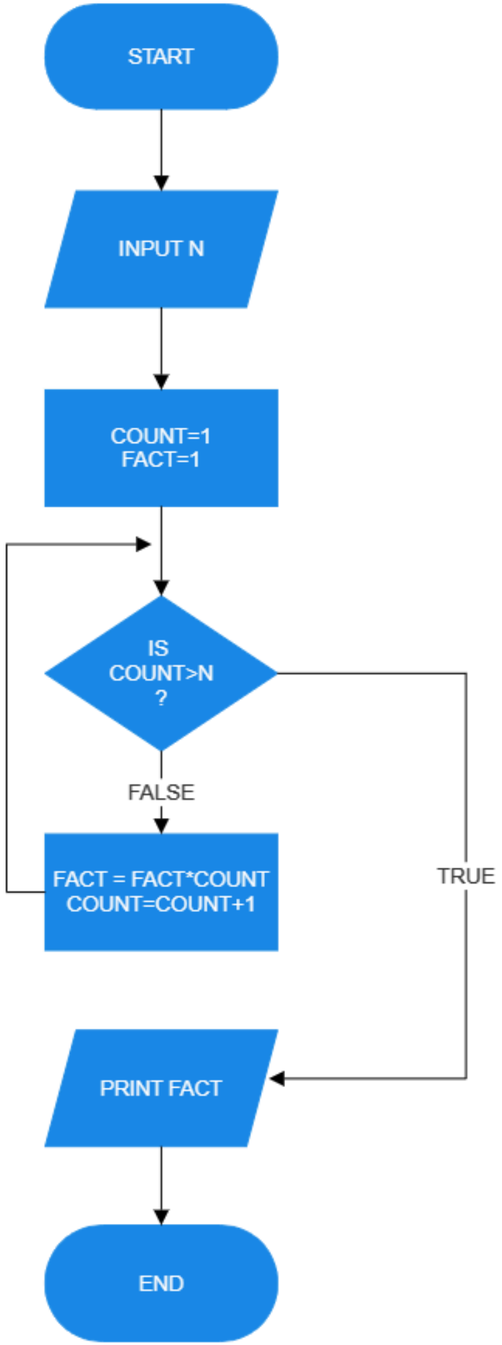
NAME OF THE ALGORITHM: COMPUTE THE FACTORIAL OF A NUMBER

Step1: START
Step 2: Input N
Step 3: fact <-1
Step 4: for count=1 to N in step of 1 do
begin
fact<-fact*count
end
Step 5: Print ‘fact of N=‘, fact
Step 6: [End of algorithm] STOP

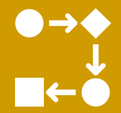
Flowchart Symbols

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectagle represents a process
	Decision	A diamond indicates a decision

FLOWCHART to find the factorial of a number



Advantages



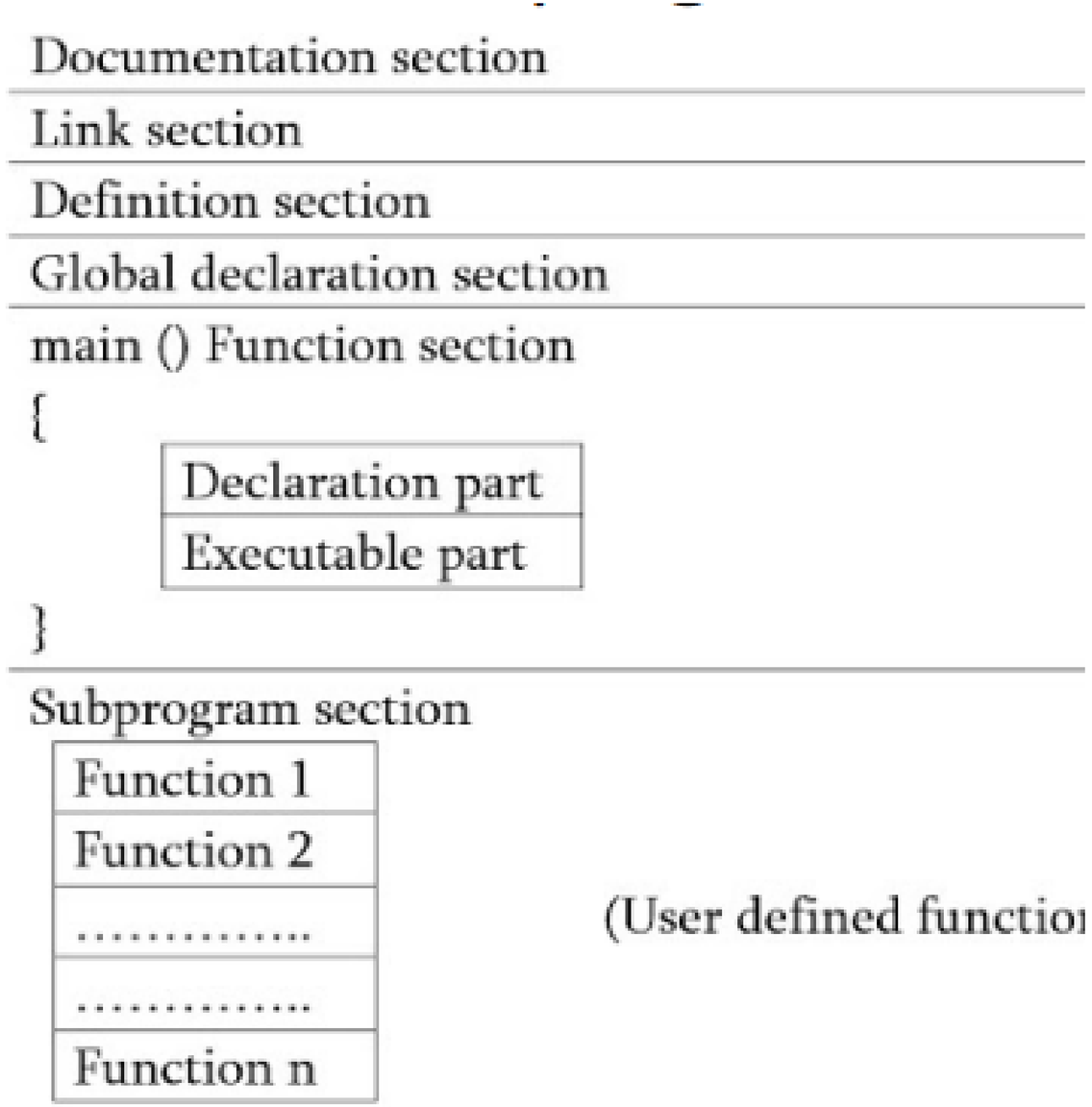
Clarity: Provides a clear visualization of the steps in a process

Analysis: Helps in analyzing and improving processes

Communication: Facilitates communication between team members and stakeholders

Problem Solving: Algorithms and Flowcharts are valuable for problem-solving by breaking down a problem into manageable steps.

GENERAL STRUCTURE OF A C PROGRAM



GENERAL STRUCTURE OF A C PROGRAM

- Documentation section
- Link Section
- Definition Section
- Global declaration section
- main( ) function section
- Subprogram

  Note:- The main() function is mandatory for all programs.

HELLO WORLD PROGRAM

```c
//Hello World Program
#include <stdio.h>
int main() {
    printf("Hello World!\n");
    return 0;
}
```

KEYWORDS

- **These are some reserved words in C which have predefined meaning to compiler.**
- **Keywords are not to be used as variable and constant names.**
- **All keywords have fixed meanings and these meanings cannot be changed**

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

IDENTIFIERS

Identifiers are used to name variables, functions, arrays, etc

Identifiers must begin with a letter or underscore (_) and can be followed by any combination of letters, underscores, or digits..

Keywords cannot be used as identifiers.

Identifiers are case-sensitive.

Examples of valid variable names: Int, _product, Hello7

Examples of invalid variable names: Val_$, 4sol, break

Operators

Operators perform operations on variables and values.
The different operators are:

- Arithmetic : +,-,*,/,%
- Relational: ==, !=, <, <=, >, >=
- Logical: && , ||, !
- Increment and Decrement: ++, --
- Bitwise:
  - Logical: &, |, ^
  - Shift: >> , <<
  - Complement ~
- Assignment +=, -=, /= *=
- Conditional ? :

Associativity and Precedence

Precedence:

determines the order in which different operators are evaluated in an expression when there are multiple operators. Operators with higher precedence are evaluated before operators with lower precedence. For example, in the expression a + b * c, multiplication (*) has higher precedence than addition (+), so b * c is evaluated first.

Associativity:

refers to the direction in which an expression is evaluated when multiple operators of the same precedence appear in an expression. In most programming languages:

- Left-associative operators are evaluated from left to right. For example, subtraction (-) is left-associative, so a - b - c is evaluated as (a - b) - c.
- Right-associative operators are evaluated from right to left. For example, assignment (=) is right-associative, so a = b = c is evaluated as a = (b = c)

Precedence	Operator	Description	Associativity
1 highest	::	Scope resolution	None
2	++ -- () [] . ->	Suffix increment Suffix decrement Parentheses (Function call) Brackets (Array subscripting) Element selection by reference Element selection through pointer	Left-to-right
3	++ -- + - ! ~ (*type*) * & sizeof	Prefix increment Prefix decrement Unary plus Unary minus Logical NOT Bitwise NOT (One's Complement) Type cast Indirection (dereference) Address-of Size-of	Right-to-left
4	.* ->*	Pointer to member Pointer to member	Left-to-right
5	* / %	Multiplication Division Modulo (remainder)	Left-to-right
6	+ -	Addition Subtraction	Left-to-right
7	<< >>	Bitwise left shift Bitwise right shift	Left-to-right
8	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left-to-right
9	== !=	Equal to Not equal to	Left-to-right
10	&	Bitwise AND	Left-to-right
11	^	Bitwise XOR (exclusive or)	Left-to-right
12	|	Bitwise OR (inclusive or)	Left-to-right
13	&&	Logical AND	Left-to-right
14	||	Logical OR	Left-to-right
15	?:	Ternary conditional	Right-to-left
16	= += -= *= /= %= <<= >>= &= ^= |=	Direct assignment Assignment by sum Assignment by difference Assignment by product Assignment by quotient Assignment by remainder Assignment by bitwise left shift Assignment by bitwise right shift Assignment by bitwise AND Assignment by bitwise XOR Assignment by bitwise OR	Right-to-left
17 lowest	,	Comma	Left-to-right

Problem:

Define precedence and associativity of operators in C. Resolve the below expression stepwise to obtain the final value of x. (Show all intermediate steps)

int x=(-10*(2-3)/15%(5+7)+8*6/12)

Answer:

-10 * (2 – 3) / 15 % (5 + 7) + 8 * 6 / 12

= -10 * (-1) / 15 % 12 + 8 * 6 / 12

= 10 / 15 % 12 + 8 * 6 / 12

= 0 % 12 + 8 * 6 / 12

= 0 + 48 / 12

= 0 + 4

= 4

PRIMARY DATA TYPES

Integer

- int :- 16
- short :- 8
- long :- 32
- unsigned and signed

char

Floating Point

- float :- 32 bits
- double:- 64 bits
- long double:- 80 bits

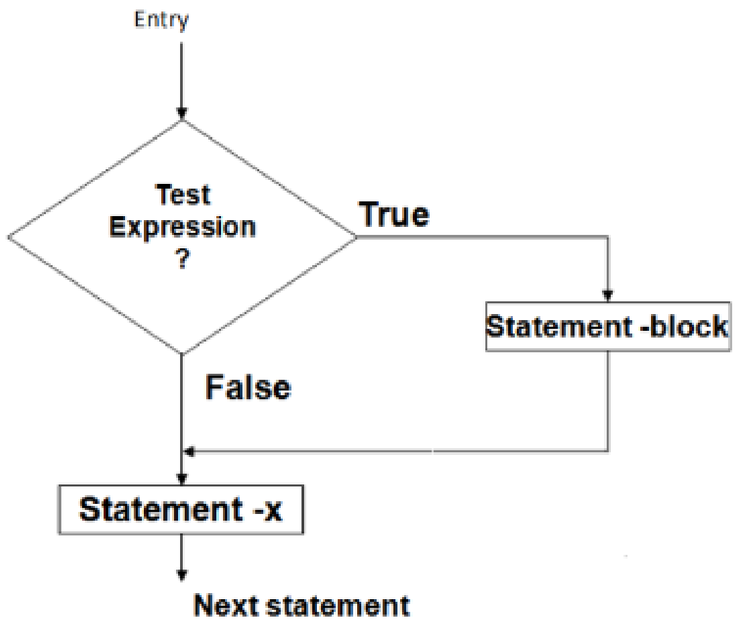
void

Decision Making, Branching & Switch

- The if Statement
- The if-else Statement
- Nested if Statements
- The else if Ladder
- The switch Statement

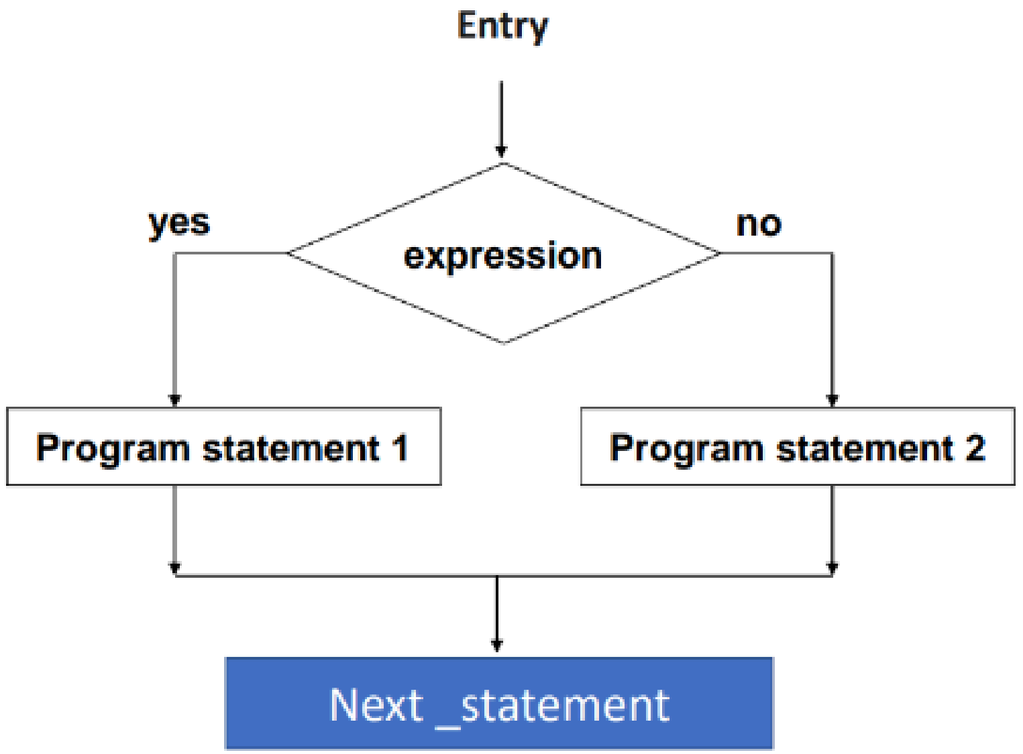
The if Statement

```
if (test Expression)
{
statement-block;
}
next_statement
```



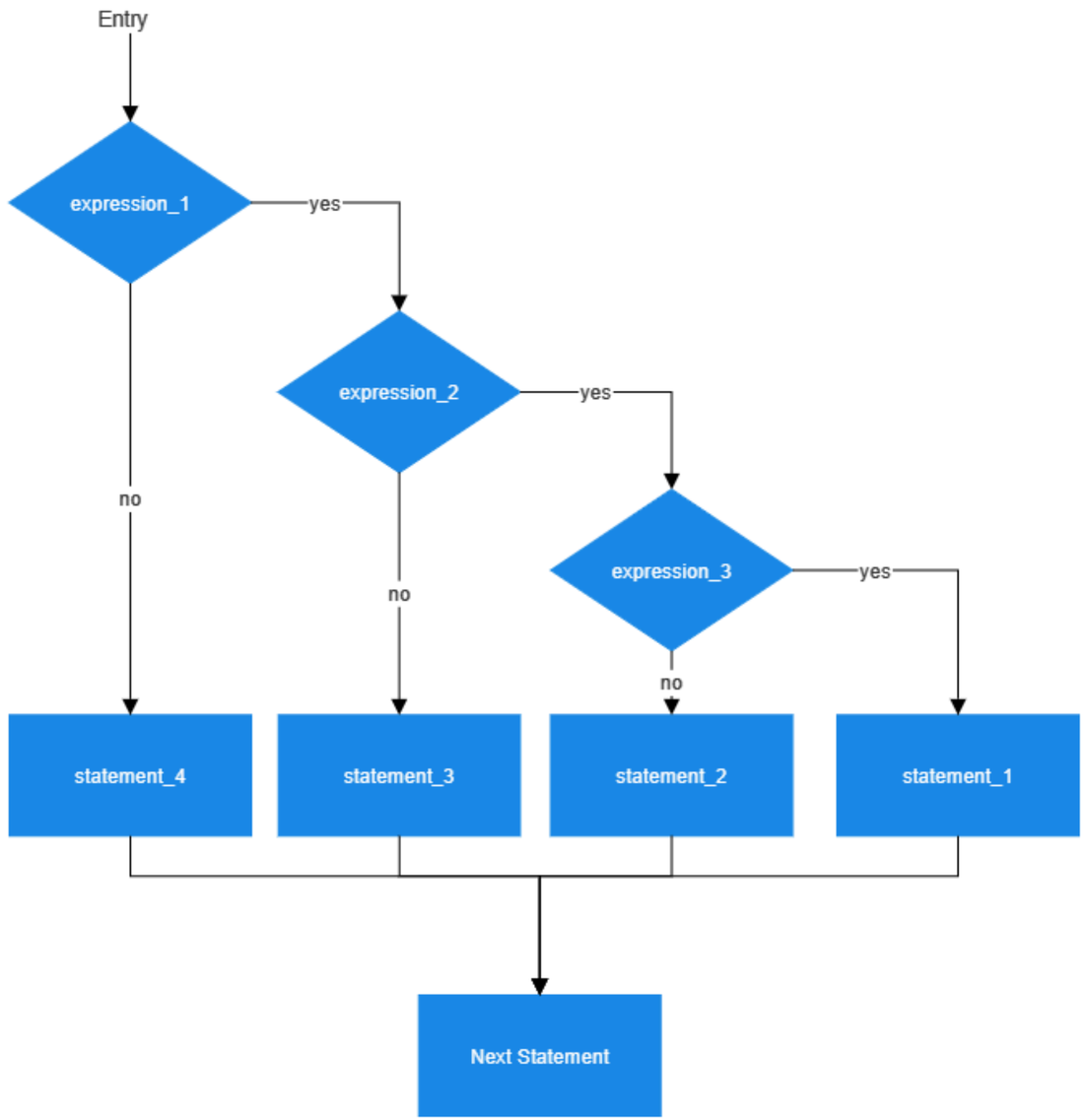
The if-else statement

```
if (test expression )
{
statement _block1
}
else
{
statement _block2
}
Next_statement
```

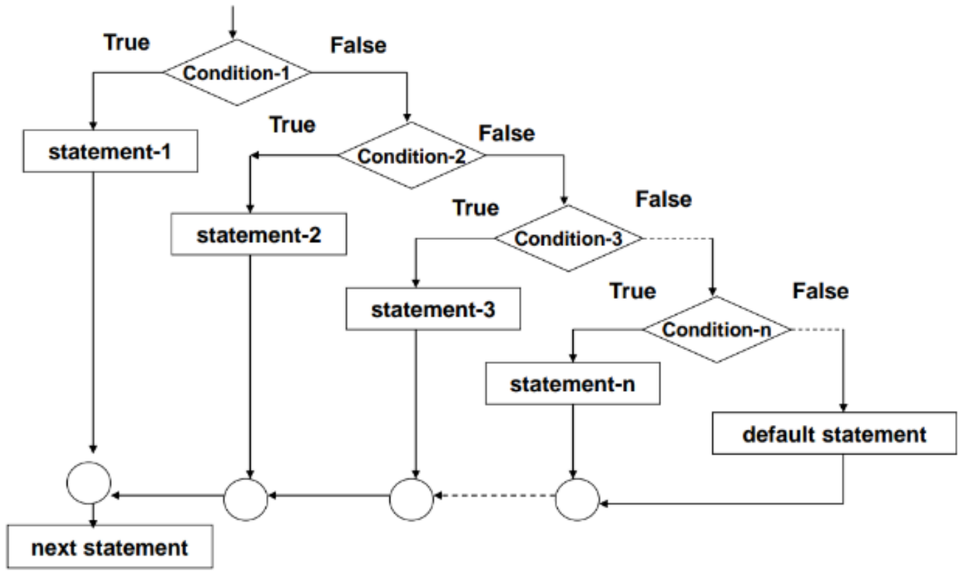


Nested if-else Statement

```
if (expression_1)
    if(expression_2)
        if(expression_3)
            statement_1;
        else
            statement_2;
    else
        statement_3;
else
    statement _block2

Next_statement
```



The else-if ladder




```
if (Expression_1 ) {
statement _block_1
}
else if (Expression_2) {
statement _block_2
}
…….
else if (Expression_n
)
{
statement _block_n
}
else {
last_statement
}
Next_statement
```

Problem:

Write a C program that determines whether a given integer is odd or even

Problem:

Write a C program that finds the largest of two given integers

Problem:

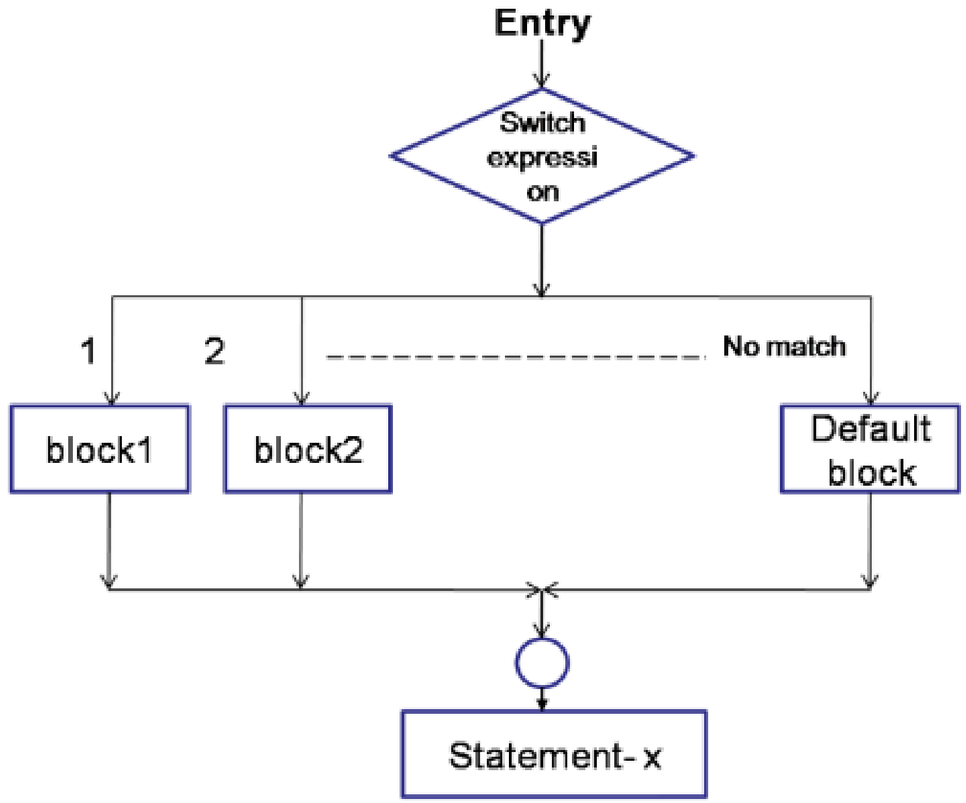
Write a C program that categorizes a given character as either a lowercase letter, an uppercase letter, a digit, or a special character.

Problem:

Write a C program to solve a quadratic equation of the form ax2+bx+c

The switch statement

```
switch ( expression )
{
case value1:
program statement(s)
break;
case value2:
program statement(s)
...
break;
case value n:
program statement(s)
break;
default:
program statement(s)
}
```



Problem:

Write a C program that uses a switch statement to handle user input for a simple choice menu between saying yes and no.

Problem:

Write a C program that performs basic arithmetic operations based on user input. using a switch statement (basically a simple calculator)

Problem:

Write a C program that determines if a given character is a vowel.

Problem:

Write a C program that assigns a letter grade based on a given numerical mark that ranges in values of 10s for eg. 10, 20, 30 out of 100 anything above 80 is an A

Loop Control Structures

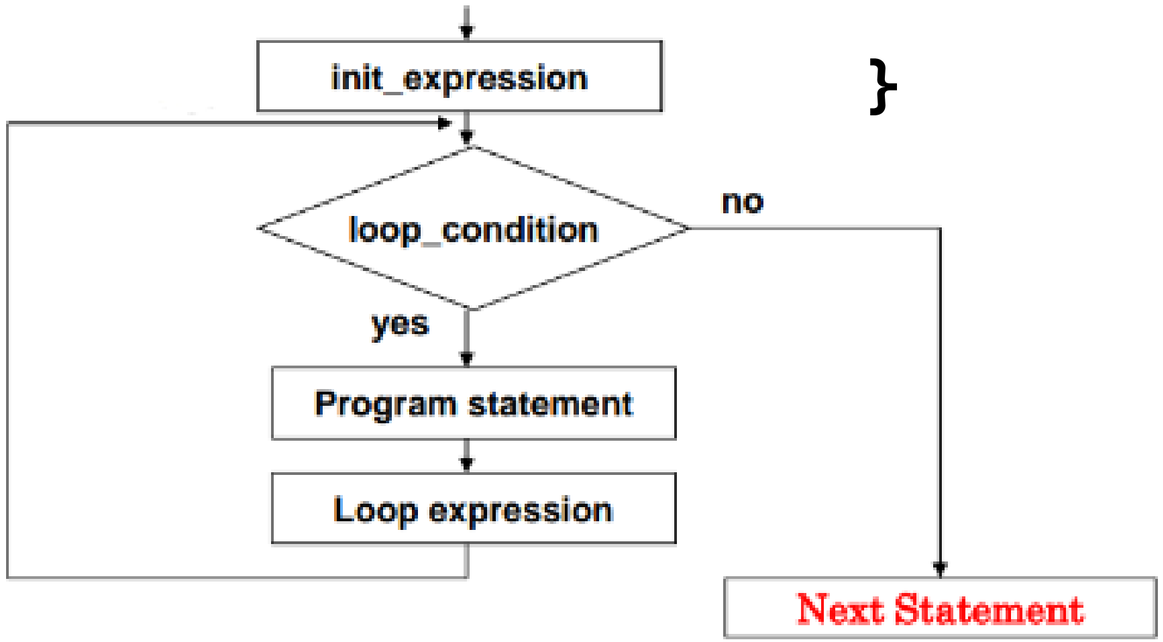
for

while

do-while

for Loop

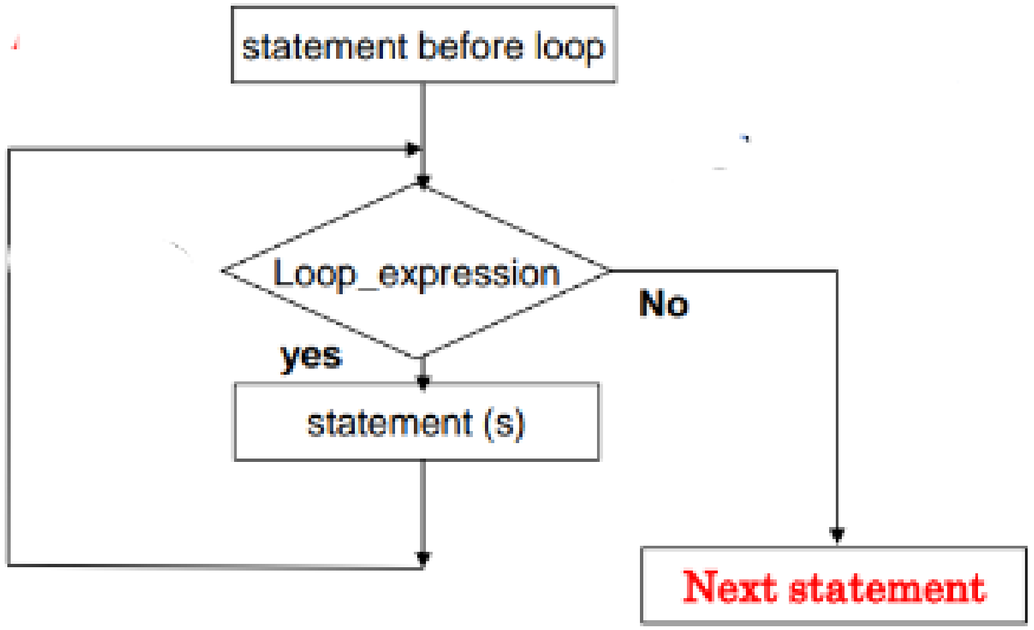
```
for(init_expression;loop_condition;loop_expre
ssion)
{

program statement(s)

}
```



Problem:

Write a C program that calculates the factorial of a given integer using a for loop

While Loop



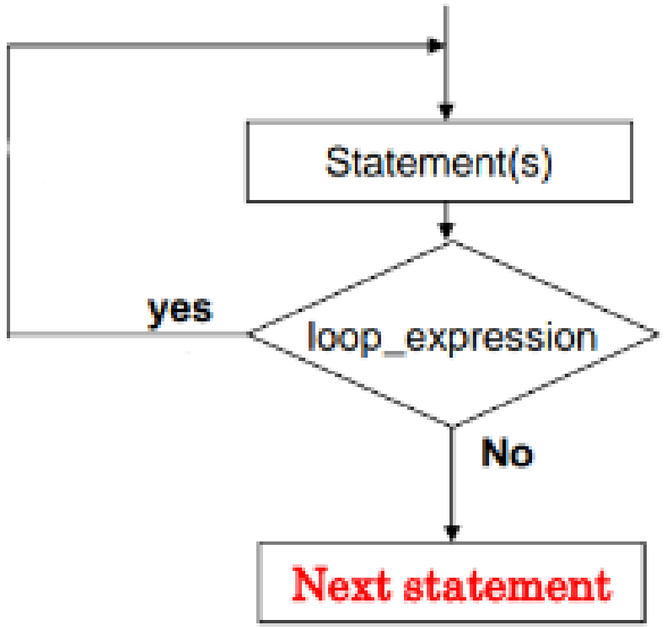

```
while ( expression )
{

program statement(s)

}
```

Problem:

Write a C program to reverse a number..

do-while loop

```
do {

program statement (s) }

while (loop_expression)
```

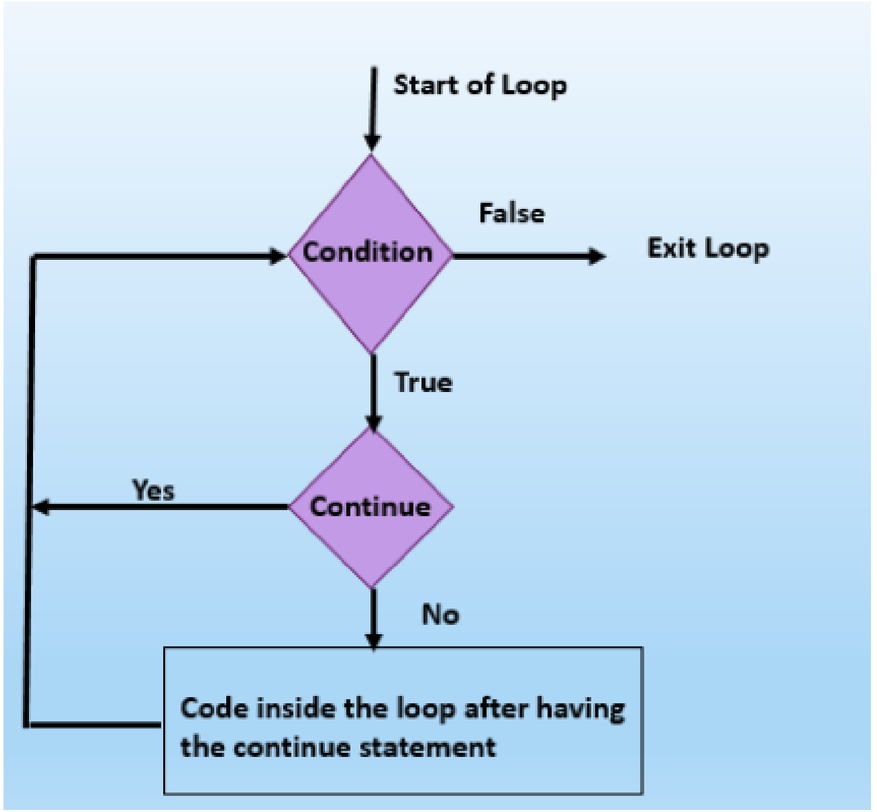


Problem:

Write a C program to reverse a number..

Continue

```
Looping Structure (loop_expression) {

    if (condition_to_continue) {
        continue;
    }

}
```

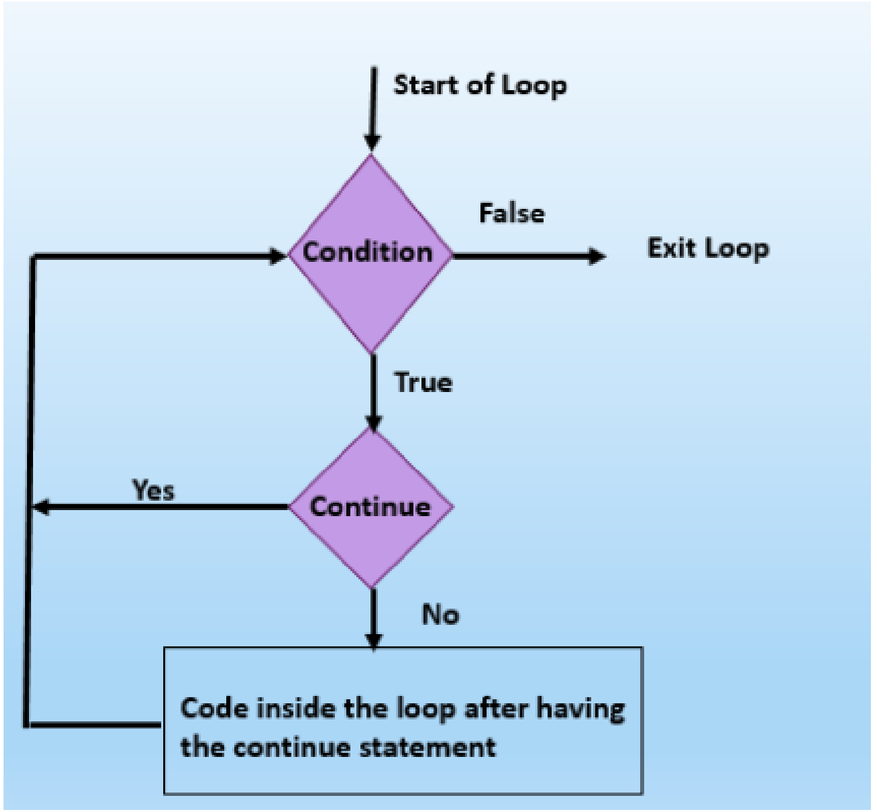


Problem:

Print all the numbers in a certain range from 0 to etc however skip even numbers.

Break

```
Looping Structure (loop_expression) {

    if (condition_to_continue) {
        break;
    }

}
```

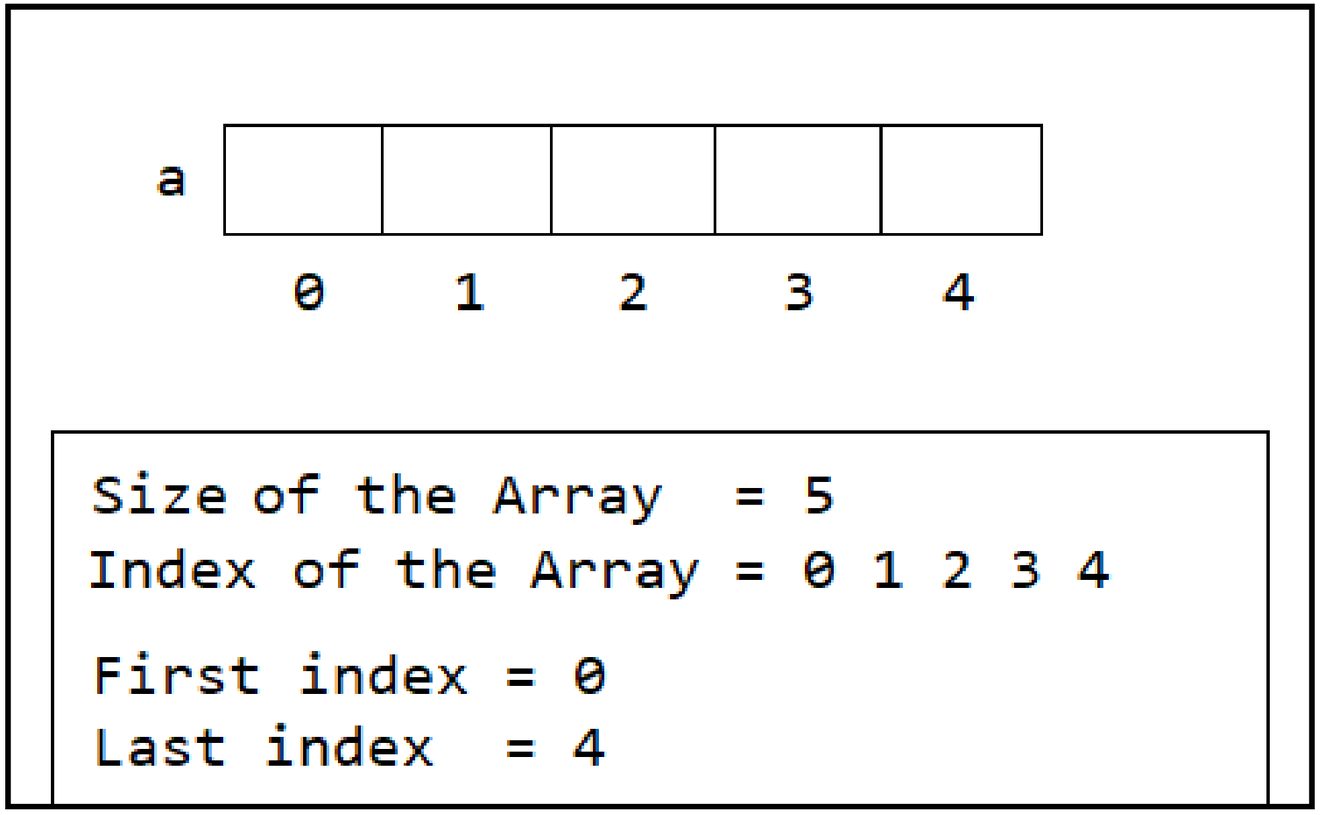


Problem:

Check if a given number is prime.

1-D Arrays

- An array is a group of related data items that share a common name.
- The array elements are placed in contiguous memory locations.
- A particular value in an array is indicated by writing an integer number called index number or subscript in square brackets after the array name.
- The least value that an index can take in array is 0..



Declaration of Array in c->
that holds integers.

int arr[5];
This will declare an array whose indexes range from 0 - 4

Now to access elements in the array to set them we can do
int value = 1
arr[0] = value
this stores the value of 1 in the index 0 of the array which is the first element

Problem:

Write a C program that reads a user-defined number of integers into an array and then prints the entered integers.

Problem:

Write a programme to Reverse an array. only use 1 Array.

Problem:

Write a programme to Delete an element from an Array.

Problem:

Write a programme to insert an element into an Array with data already present

Searching Techniques

Linear Search

- The Linear Search is applied on the set of items that are not arranged in any particular order
- In linear search , the searching process starts from the first item.
- The searching is continued till either the item is found or the end of the list is reached indicating that the item is not found.
- The items in the list are assumed to be unique

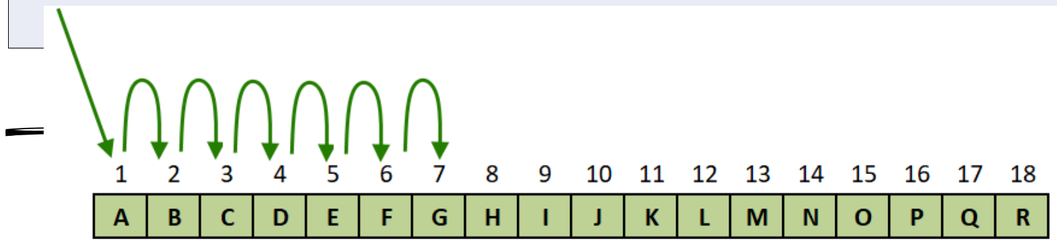
Searching Techniques

Binary Search

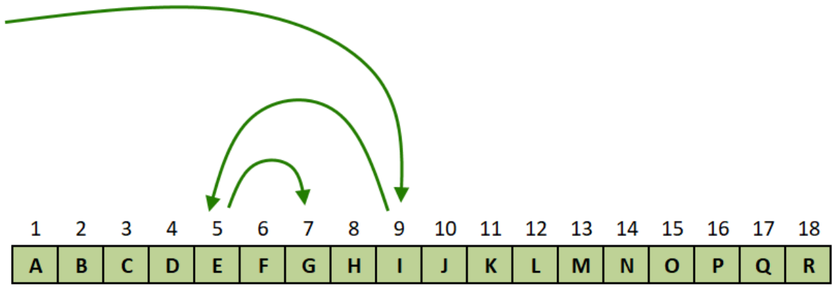
- Binary search is a divide-and-conquer algorithm used for efficiently searching a sorted array or list. It repeatedly divides the search space in half.
- Binary search requires the input data to be sorted. This is a crucial precondition for the algorithm to work correctly..
- Begin with the entire sorted array.. Check the middle element. If it's the target, the search is done. If the target is smaller, focus on the left half. If larger, focus on the right half..
- Repeat the process on the chosen half until the target is found or the search space is empty.

Linear vs Binary

LINEAR	BINARY
Data can be in any order	Data must be sorted
Preferred for arrays of small size	Preferred for arrays of large size
Slow	Fast



Linear Search - Find 'G' in sorted list A-R



Binary Search - Find 'G' in sorted list A-R

Tip:

If you are required to sort any array of values / items just use bubble sort unless explicitly mentioned

Sorting Techniques

BUBBLE SORT

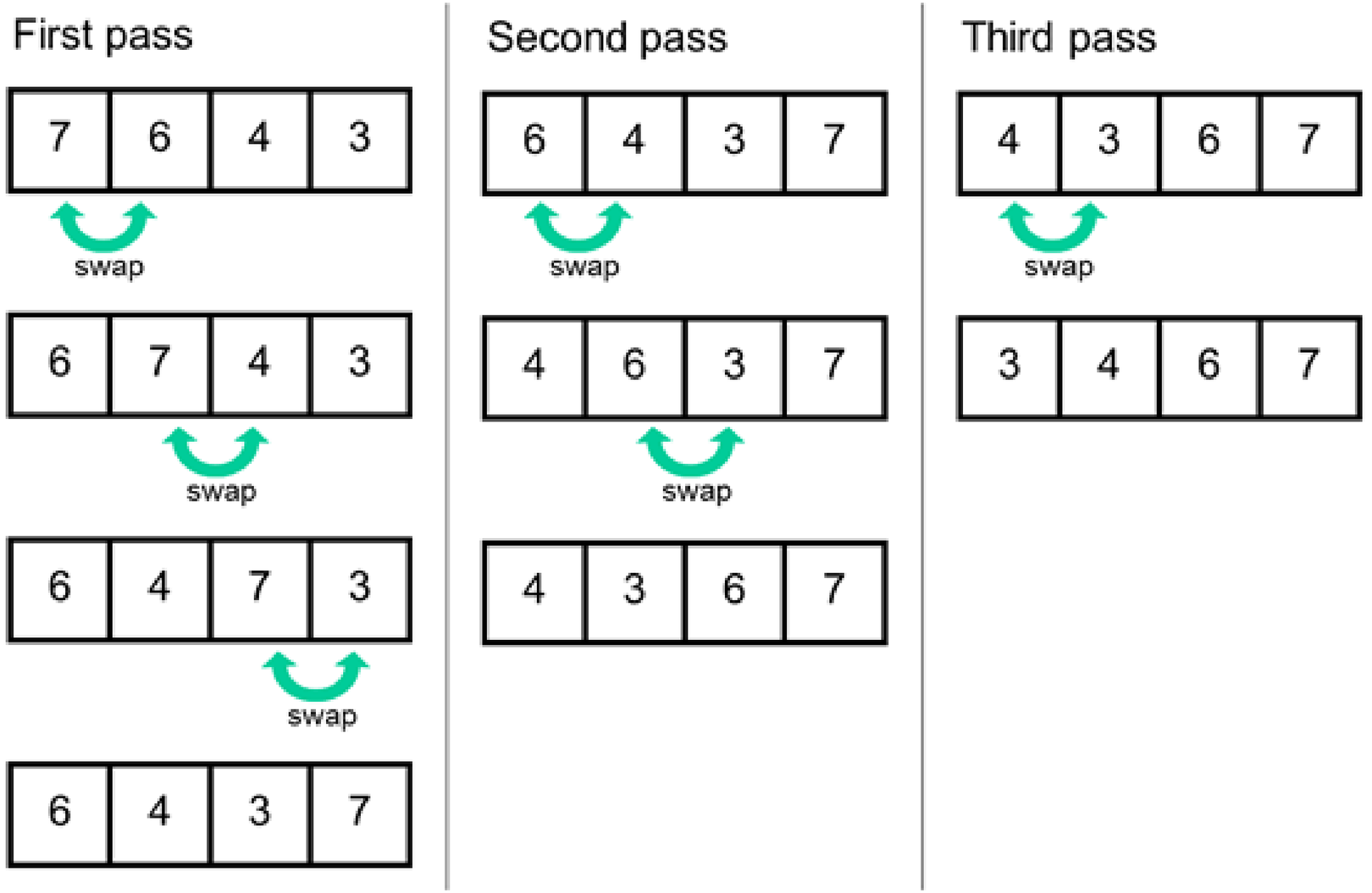
- Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order..
- It iterates through the list multiple times, and in each pass, the largest unsorted element "bubbles up" to its correct position at the end of the list.
- The process is repeated until the entire list is sorted. On each pass, the algorithm compares and swaps elements, gradually moving the largest unsorted element to its correct position.

Sorting Techniques

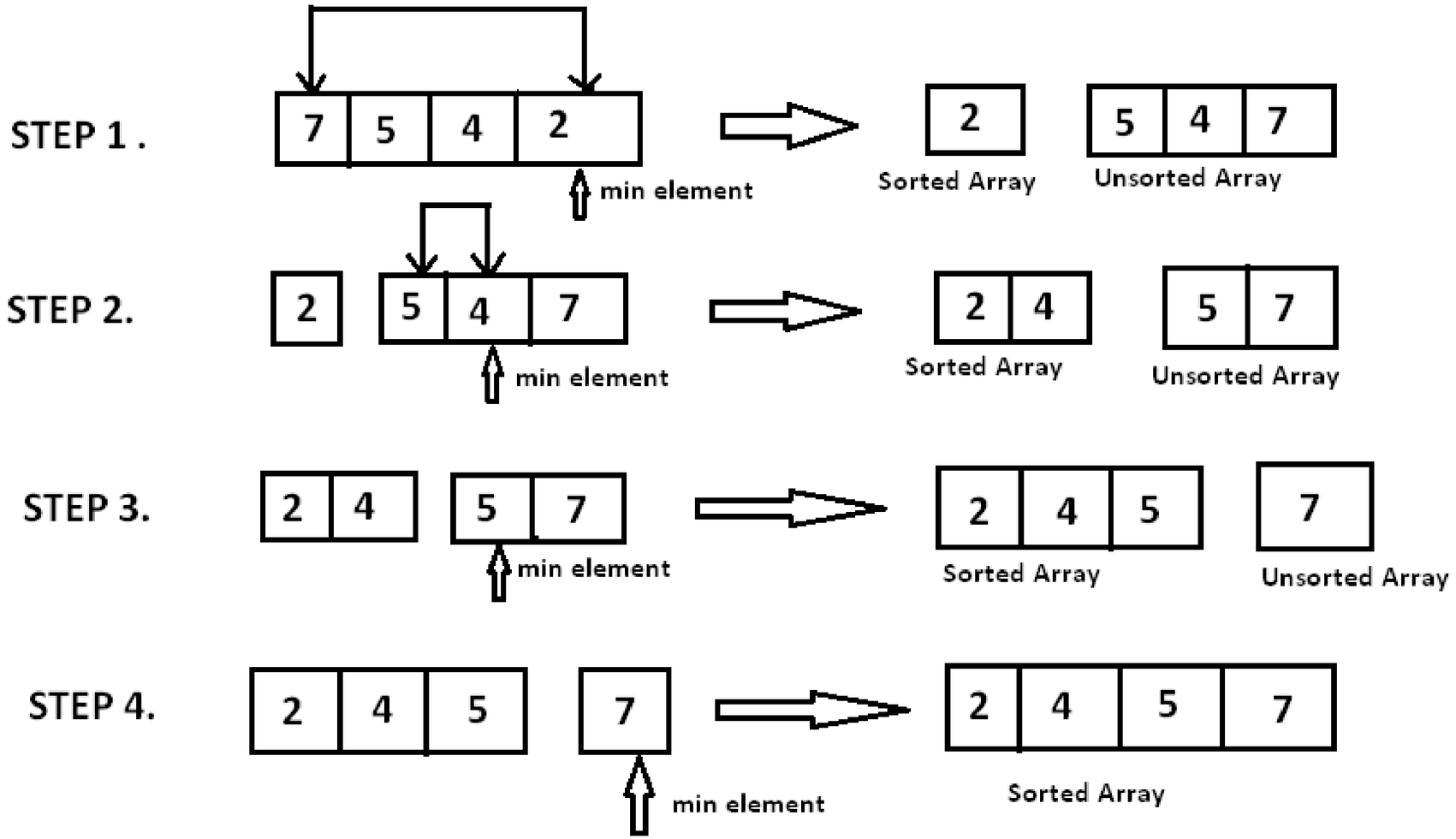
SELCTION SORT

- Selection sort starts by dividing the input list into two parts: a sorted and an unsorted region. It repeatedly selects the minimum element from the unsorted region and swaps it with the first element of the unsorted region.
- The algorithm iterates through the unsorted region, finding the minimum element and moving it to the sorted region. This process continues until the entire list is sorted.
- Selection sort minimizes the number of swaps by only making a single swap for each pass through the unsorted region.

Bubble Sort Illustration



Selection Sort Illustration



IECSE MEMBERSHIP DRIVE

2-D Arrays

It is an ordered table of homogeneous elements.

It can be imagined as a two dimensional table made of elements, all of them of a same uniform data type.

It is generally referred to as matrix, of some rows and some columns.

It is also called as a two-subscripted variable.

Problem:

Write a programme to enter and print an 2d array

Problem:

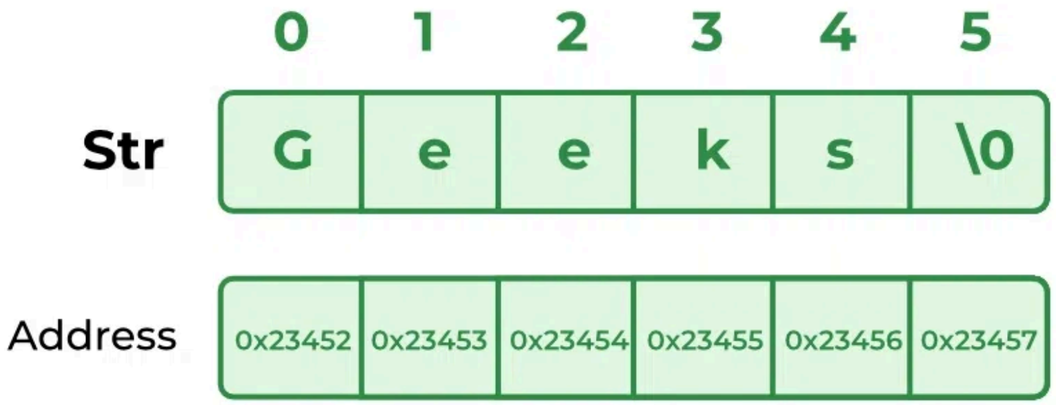
Write a programme for Matrix Multiplication.

Strings

A string is an array of characters.

Any group of characters (except double quote sign) defined between double quotation marks is a constant string.

Character strings are often used to build meaningful and readable programs.



Problem:

Write a C program that performs the following tasks:

1. Initialization of string
2. enter a string.
3. Print the entered string

Problem:

Write a C program that checks wether a given string is a palindrome (Palindrome is where the reverse of a object is the same as the origin object)

Built-in String Handling functions

- Used to manipulate a given string.
- These functions are part of “string.h” header file.
  - strlen ():- gives the length of the string.
  - strcpy ():-copies one string to other.
  - strcmp ():-compares the two strings.
  - strcat ():- Concatinate the two strings

Strlen()

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "Hello, World!";
    int length = strlen(str);
    printf("\n");
    printf("Length of the string: %d\n", length);
    printf("\n");
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\arygu\Aryan's files\vscode files\IECSE_PSUC_Workshop\Part5> cd "c:
ngOperations.c -o StringOperations } ; if ($?) { .\StringOperations }

Length of the string: 13

PS C:\Users\arygu\Aryan's files\vscode files\IECSE_PSUC_Workshop\Part5>
```

Strcpy()

```c
#include <stdio.h>
#include <string.h>

int main() {
    char src[100] = "Copy this string";
    char dest[100];

    strcpy(dest, src);
    printf("\n");
    printf("Destination string: %s\n", dest);
    printf("\n");
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\arygu\Aryan's files\vscode files\IECSE_PSUC_Workshop\Part5> cd "c:\
ngOperations.c -o StringOperations } ; if ($?) { .\StringOperations }

Destination string: Copy this string
```

Strcmp()

```c
#include <stdio.h>
#include <string.h>

int main() {
    char str1[] = "apple";
    char str2[] = "banana";

    printf("\n");

    int result = strcmp(str1, str2);
    if (result < 0) {
        printf("'%s' comes before '%s' lexicographically.\n", str1, str2);
    } else if (result > 0) {
        printf("'%s' comes after '%s' lexicographically.\n", str1, str2);
    } else {
        printf("'%s' is lexicographically equal to '%s'.\n", str1, str2);
    }

    printf("\n");

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\arygu\Aryan's files\vscode files\IECSE_PSUC_Workshop\Part5> cd "c:\Users\arygu\Aryan's files\vscode
ngOperations.c -o StringOperations } ; if ($?) { .\StringOperations }

'apple' comes before 'banana' lexicographically.
```

Strcat()

```c
#include <stdio.h>
#include <string.h>

int main() {
    char dest[50] = "Hello, ";
    char src[] = "World!";

    strcat(dest, src);
    printf("\n");
    printf("Concatenated string: %s\n", dest);
    printf("\n");
    return 0;
}
```

```
LEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

C:\Users\arygu\Aryan's files\vscode files\IECSE_PSUC_Workshop\Part5> cd '
perations.c -o StringOperations } ; if ($?) { .\StringOperations }

catenated string: Hello, World!

C:\Users\arygu\Aryan's files\vscode files\IECSE_PSUC_Workshop\Part5>
```

Problem:

Write a C program that checks the amount of words in a sentence.

Problem:

Write a C program that SortsTheNames in an Array.

Problems you may get?:

Write a menu driven C program to perform the following tasks using switch statement (Note: The program should continue till the user inputs Exit option)

1.Reverse of a given number

2.Sum of digits

3. Exit. (3)

Problems you may get?:

Write a C program to read a 1D array, remove prime numbers, and display the modified array in ascending order.

Problems you may get?:

Create a 2D array where each row corresponds to a student, and each column represents a subject. Each cell in the array holds the marks of a student of a particular subject. Develop a C program to compute the average marks of each student and the average marks of each subject. (Inputs are read from the user)

Problems you may get? (Final Boss):

Write a C program that allows the user to input a string, a substring to find, and another substring to replace it with. The program should search for occurrences of the given substring within the input string and replace each occurrence.

All The Very Best