USING

{C}
Programming

Introduction to computer and programming

What is Computer?

- The word computer comes from the word “compute”, which means, “to calculate”.
- A computer is an electronic device that can perform arithmetic operations at high speed and it can process data, pictures, sound and graphics.
- It can solve highly complicated problems quickly and accurately.

Advantages of Computer

- Speed
  - It can calculate millions of expression within a fraction of second.
- Storage
  - It can store large amount of data using various storage devices.
- Accuracy
  - It can perform the computations at very high speed without any mistake.
- Reliability
  - The information stored in computer is available after years in same form. It works 24 hours without any problem as it does not feel tiredness.
- Automation
  - Once the task is created in computer, it can be repeatedly performed again by a single click whenever we want.
- Multitasking

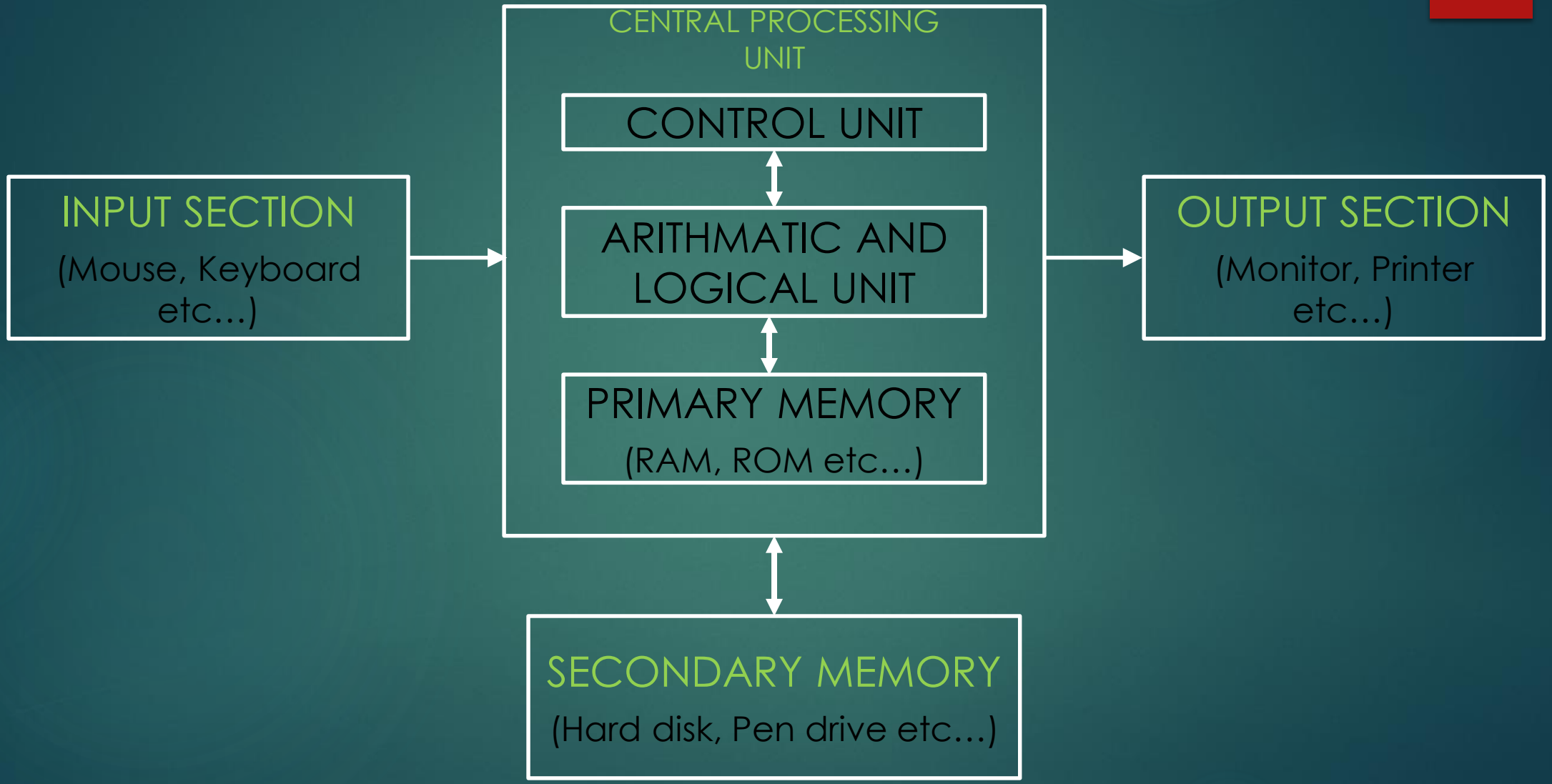
Disadvantages of Computer

- Lake of intelligence
  - It can not think while doing work.
  - It does not have natural intelligence.
  - It can not think about properness, correctness or effect of work it is doing.
- Unable to correct mistake
  - It can not correct mistake by itself.
  - So if we provide wrong or incorrect data then it produces wrong result or perform wrong calculations.

Block Diagram of Computer

- It is a pictorial representation of a computer which shows how it works inside.
- It shows how computer works from feeding/inputting the data to getting the result.

Block Diagram of Computer



Block diagram of computer (Input Section)

- The devices used to enter data in to computer system are called input devices.
- It converts human understandable input to computer controllable data.
- CPU accepts information from user through input devices.
- Examples: Mouse, Keyboard, Touch screen, Joystick etc…

Block diagram of computer (Output Section)

- The devices used to send the information to the outside world from the computer is called output devices.
- It converts data stored in 1s and 0s in computer to human understandable information.
- Examples: Monitor, Printer, Plotter, Speakers etc…

Block diagram of computer (Central Processing Unit (CPU))

- It contains electronics circuit that processes the data based on instructions.
- It also controls the flow of data in the system.
- It is also known as brain of the computer.
- CPU consists of,
  - Arithmetic Logic Unit (ALU)
    - It performs all arithmetic calculations such as add, subtract, multiply, compare, etc. and takes logical decision.
    - It takes data from memory unit and returns data to memory unit, generally primary memory (RAM).
  - Control Unit (CU)
    - It controls all other units in the computer system. It manages all operations such as reads instruction and data from memory.
  - Primary Memory
    - It is also known as main memory.

Block diagram of computer (Secondary Memory)

- Secondary memory is also called Auxiliary memory or External memory.
- It is Used to store data permanently.
- It can be modified easily.
- It can store large data compared to primary memory. Now days, it is available in Terabytes.
- Examples: Hard disk, Floppy disk, CD, DVD, Pen drive, etc…

What is Hardware?

- Hardware refers to the physical parts of a computer.
- The term hardware also refers to mechanical device that makes up computer.
- User can see and touch the hardware components.
- Examples of hardware are CPU, keyboard, mouse, hard disk, etc…

What is Software?

- A **set of instruction** in a logical order **to perform a meaningful task** is called program and **a set of program** is called software.
- It tell the hardware how to perform a task.
- Types of software
  - System software
    - It is designed to operate the computer hardware efficiently.
    - Provides and maintains a platform for running application software.
    - Examples: Windows, Linux, Unix etc.
  - Application software
    - It is designed to help the user to perform general task such as word processing, web browser etc.
    - Examples: Microsoft Word, Excel, PowerPoint etc.

Categories of System Software

- Operating system
  - It controls hardware as well as interacts with users, and provides different services to user.
  - It is a bridge between computer hardware and user.
  - Examples: Windows XP, Linux, UNIX, etc…
- System support software
  - It makes working of hardware more efficiently.
  - For example drivers of the I/O devices or routine for socket programming, etc…
- System development software
  - It provides programming development environment to programmers.
  - Example: Editor, pre-processor, compiler, interpreter, loader, etc…

Categories of Application Software

- General purpose software
  - It is used widely by many people for some common task, like word processing, web browser, excel, etc…
  - It is designed on vast concept so many people can use it.
- Special purpose software
  - It is used by limited people for some specific task like accounting software, tax calculation software, ticket booking software, banking software etc…
  - It is designed as per user's special requirement.

Compiler, Interpreter and Assembler

- Compiler translates program of higher level language to machine language. It converts whole program at a time.
- Interpreter translates program of higher level language to machine language. It converts program line by line.
- Assembler translates program of assembly language to machine language.

Types of Computer Languages

- Machine level language OR Low level language
  - It is language of 0's and 1's.
  - Computer directly understand this language.
- Assembly language
  - It uses short descriptive words (MNEMONIC) to represent each of the machine language instructions.
  - It requires a translator knows as assembler to convert assembly language into machine language so that it can be understood by the computer.
  - Examples: 8085 Instruction set
- Higher level language
  - It is a machine independent language.
  - We can write programs in English like manner and therefore easier to learn and use.
  - Examples: C, C++, JAVA etc…

Types of Computer Languages

Flowchart	Algorithm
Flowchart is a pictorial or graphical representation of a program.	Algorithm is a finite sequence of well defined steps for solving a problem.
It is drawn using various symbols	It is written in the natural language like
Easy to understand.	Difficult to understand.
Easy to show branching and looping.	Difficult to show branching and looping.
Flowchart for big problem is impractical.	Algorithm can be written for any problem.

Symbols used in Flowchart

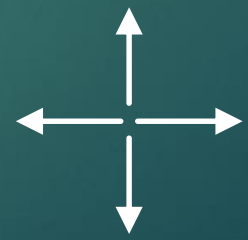
Start / Stop

Input / Output

Process

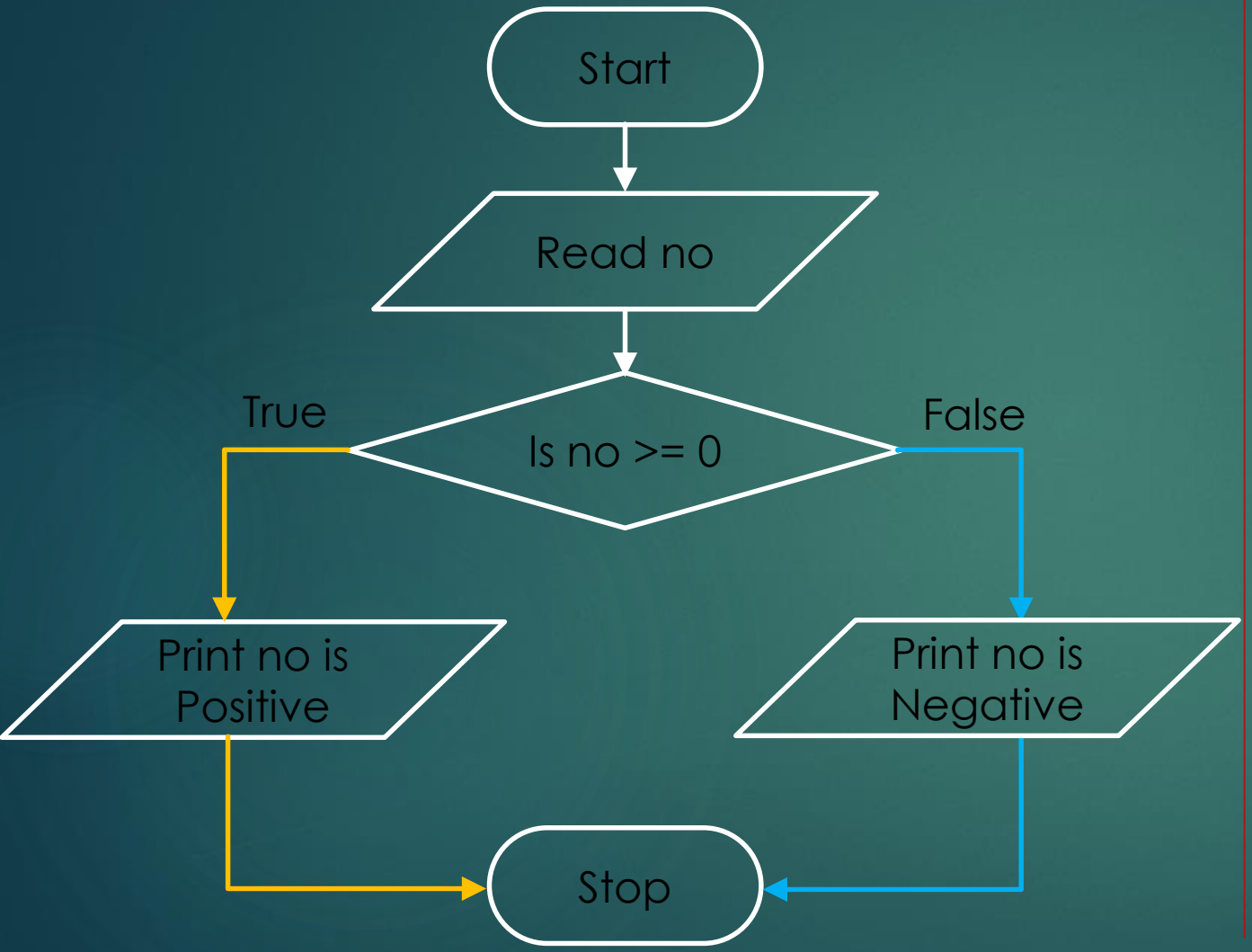
Decision Making

Subroutine



Arrows

Number is positive or negative



Step 1: Read no.

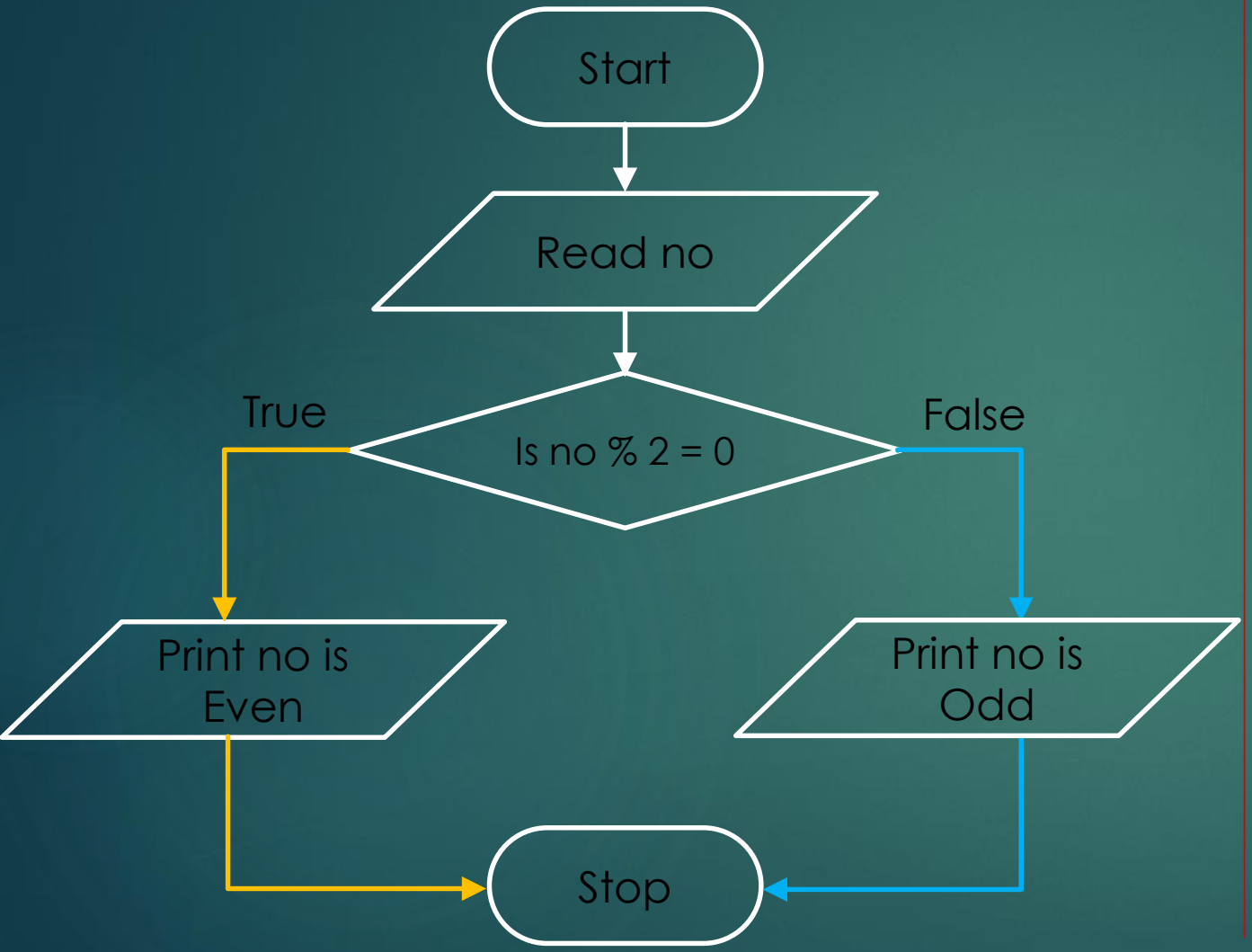
Step 2: If no is greater than equal zero, go to step 4.

Step 3: Print no is a negative number, go to step 5.

Step 4: Print no is a positive number.

Step 5: Stop.

Number is odd or even



Step 1: Read no.

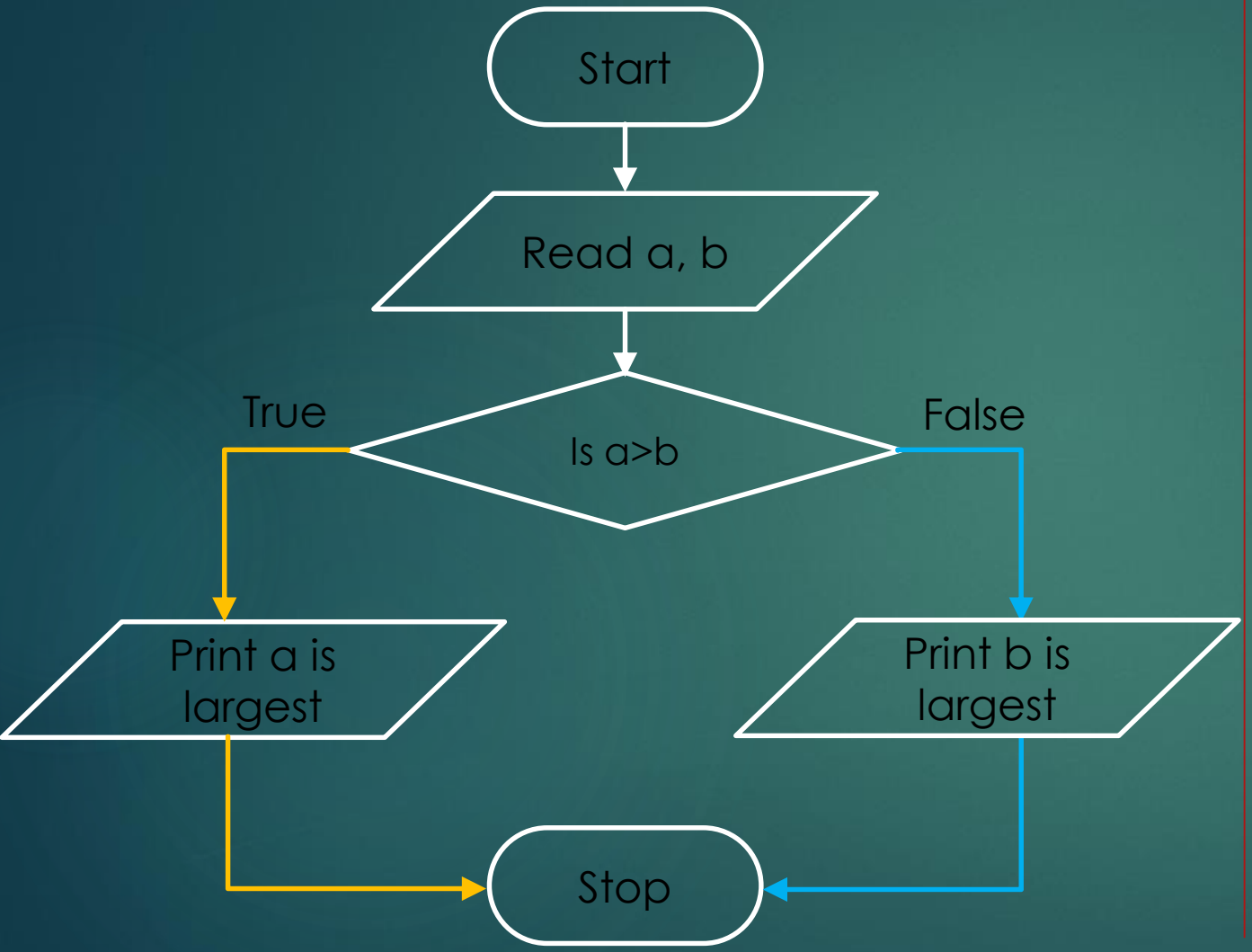
Step 2: If no mod 2 = 0, go to step 4.

Step 3: Print no is a odd, go to step 5.

Step 4: Print no is a even.

Step 5: Stop.

Largest number from 2 numbers



Step 1: Read a, b.

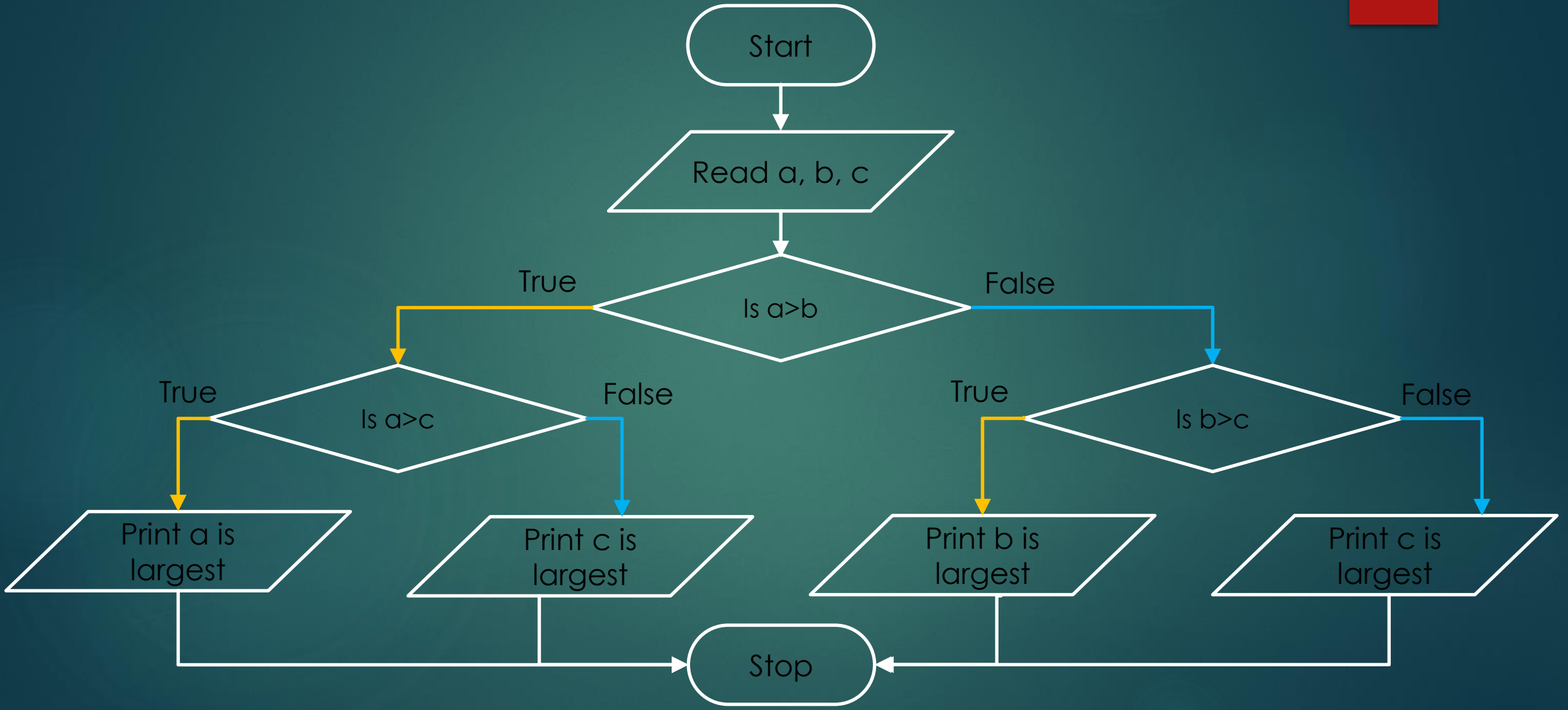
Step 2: If a>b, go to step 4.

Step 3: Print b is largest number, go to step 5.

Step 4: Print a is largest number.

Step 5: Stop.

Largest number from 3 numbers (Flowchart)



Largest number from 3 numbers (Algorithm)

Step 1: Read a, b, c.

Step 2: If a>b, go to step 5.

Step 3: If b>c, go to step 8.

Step 4: Print c is largest number, go to step 9.

Step 5: If a>c, go to step 7.

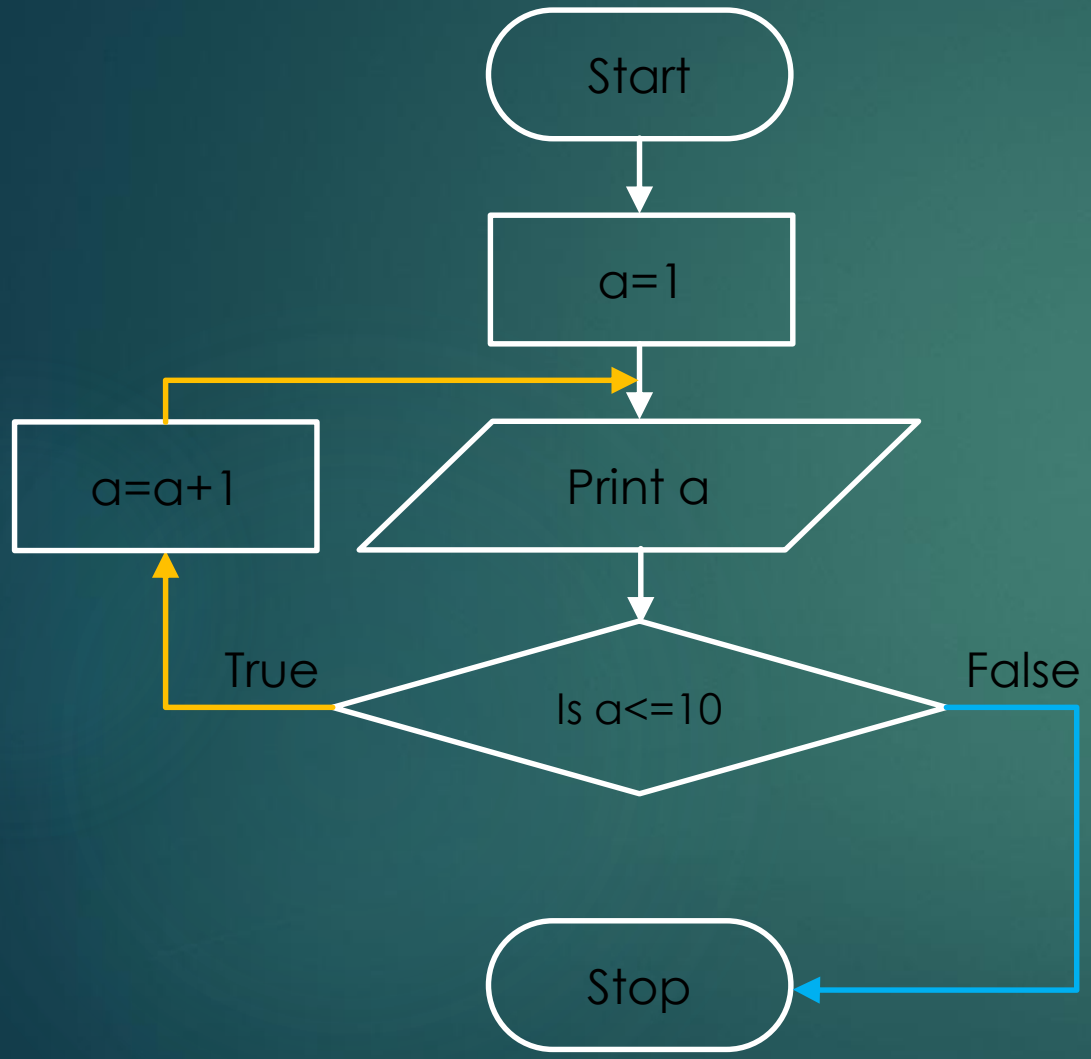
Step 6: Print c is largest number, go to step 9.

Step 7: Print a is largest number, go to step 9.

Step 8: Print b is largest number.

Step 9: Stop.

Print 1 to 10



Step 1: Initialize a to 1.

Step 2: Print a.

Step 3: Repeat step 2 until a<=10.

Step 3.1: a=a+1.

Step 4: Stop.

Thank you

USING

{C}
Programming

Fundamentals of C

Structure of C Program

```
Documentation section
        (Used for comments)
Link section

Definition section

Global declaration section
(Variables used in more than
one functions)
void main ()
{
        Declaration part
        Executable part
}
Subprogram section
        (User defined functions)
```

Program

```
1  // Program for addition of 2 nos
2
3
4
5
6  void fun();
7
8  int a=10;
9
10
11 void main( )
12  {
13   printf("Value of a inside main function: %d", a);
14   fun();
15  }
16
14 void fun()
18 {printf("Value of a inside fun function: %d", a);}
```

Comments

- A comment is an explanation or description of the source code of the program
- It helps a programmer to explain logic of the code and improves program readability.
- At run-time, a comment is ignored by the compiler.
- There are two types of comments in C:
  - Single line comment
    - Represented as // double forward slash
    - It is used to denote a single line comment only.
    - Example: `// Single line comment`
  - Multi-line comment
    - Represented as /* any_text */ start with forward slash and asterisk (/*) and end with asterisk and forward slash (*/).
    - It is used to denote single as well as multi-line comment.
    - Example:

```
/* multi line comment line -1
   multi line comment line -2 */
```

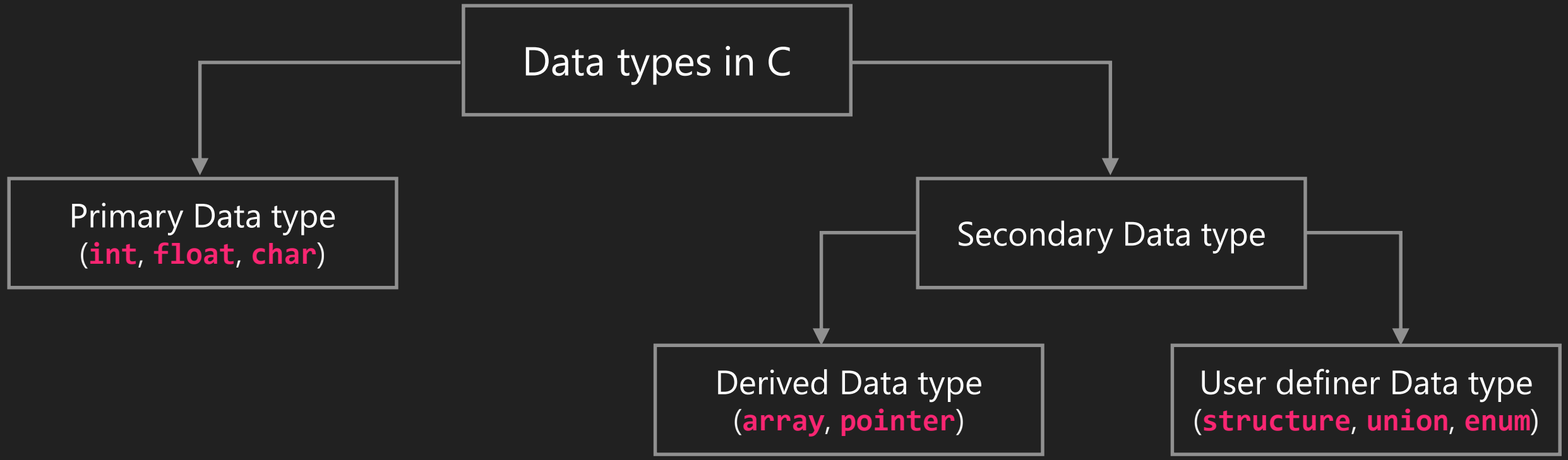
Header files

- A header file is a file with extension .h which contains the set of predefined standard library functions.
- The "#include" preprocessing directive is used to include the header files with extension in the program.

Header file	Description
stdio.h	Input/Output functions (printf and scanf)
conio.h	Console Input/Output functions (getch and clrscr)
math.h	Mathematics functions (pow, exp, sqrt etc…)
string.h	String functions (strlen, strcmp, strcat etc…)

Data Types

- Data types are defined as the data storage format that a variable can store a data.
- It determines the type and size of data associated with variables.



Primary Data Type

- Primary data types are built in data types which are directly supported by machine.
- They are also known as fundamental data types.
  - `int`:
    - `int` datatype can store integer number which is whole number without fraction part such as 10, 105 etc.
    - C language has 3 classes of integer storage namely `short int`, `int` and `long int`. All of these data types have signed and unsigned forms.
    - Example: `int` a=10;
  - `float`:
    - `float` data type can store floating point number which represents a real number with decimal point and fractional part such as 10.50, 155.25 etc.
    - When the accuracy of the floating point number is insufficient, we can use the `double` to define the number. The double is same as float but with longer precision.
    - To extend the precision further we can use `long double` which consumes 80 bits of memory space.
    - Example: `float` a=10.50;

Primary Data Type (cont…)

- **char**:
  - **Char** data type can store single character of alphabet or digit or special symbol such as ‘a’, ‘5’ etc.
  - Each character is assigned some integer value which is known as ASCII values.
  - Example: **char** a=‘a’;
- **void**:
  - The **void** type has no value therefore we cannot declare it as variable as we did in case of **int** or **float** or **char**.
  - The **void** data type is used to indicate that function is not returning anything.

Secondary Data Type

- Secondary data types are not directly supported by the machine.
- It is combination of primary data types to handle real life data in more convenient way.
- It can be further divided in two categories,
  - Derived data types: Derived data type is extension of primary data type. It is built-in system and its structure cannot be changed. Examples: Array and Pointer.
    - Array: An array is a fixed-size sequenced collection of elements of the same data type.
    - Pointer: Pointer is a special variable which contains memory address of another variable.
  - User defined data types: User defined data type can be created by programmer using combination of primary data type and/or derived data type. Examples: Structure, Union, Enum.
    - Structure: Structure is a collection of logically related data items of different data types grouped together under a single name.
    - Union: Union is like a structure, except that each element shares the common memory.
    - Enum: Enum is used to assign names to integral constants, the names make a program easy to read and maintain.

Variables and Constants

- Variable is a symbolic name given to some value which can be changed.
- x, y, a, $count$, $etc.$ can be variable names.
- $x=5 \quad a=b+c$
- Constant is a fixed value which cannot be changed.
- 5, −7.5, 1452, 0, 3.14, $etc.$

Tokens

- The smallest individual unit of a program is known as token.
- C has the following tokens:
  - Keywords
    - C reserves a set of 32 words for its own use. These words are called keywords (or reserved words), and each of these keywords has a special meaning within the C language.
  - Identifiers
    - Identifiers are names that are given to various user defined program elements, such as variable, function and arrays.
  - Constants
    - Constants refer to fixed values that do not change during execution of program.
  - Strings
    - A string is a sequence of characters terminated with a null character \0.
  - Special Symbols
    - Symbols such as #, &, =, * are used in C for some specific function are called as special symbols.
  - Operators
    - An operator is a symbol that tells the compiler to perform certain mathematical or logical operation.

Operators

- Arithmetic operators (+, - , *, /, %)
- Relational operators (<, <=, >, >=, ==, !=)
- Logical operators (&&, ||, !)
- Assignment operators (+=, -=, *=, /=)
- Increment and decrement operators (++, --)
- Conditional operators (?:)
- Bitwise operators (&, |, ^, <<, >>)
- Special operators ()

Arithmetic Operators

- Arithmetic operators are used for mathematical calculation.

Operator	Meaning	Example	Description
+	Addition	a + b	Addition of a and b
-	Subtraction	a – b	Subtraction of b from a
*	Multiplication	a * b	Multiplication of a and b
/	Division	a / b	Division of a by b
%	Modulo division- remainder	a % b	Modulo of a by b

Relational Operators

- Relational operators are used to compare two numbers and taking decisions based on their relation.
- Relational expressions are used in decision statements such as if, for, while, etc…

Operator	Meaning	Example	Description
<	Is less than	a < b	a is less than b
<=	Is less than or equal to	a <= b	a is less than or equal to b
>	Is greater than	a > b	a is greater than b
>=	Is greater than or equal to	a >= b	a is greater than or equal to b
==	Is equal to	a = b	a is equal to b
!=	Is not equal to	a != b	a is not equal to b

Logical Operators

- Logical operators are used to test more than one condition and make decisions.

Operator	Meaning
&&	logical AND (Both non zero then true, either is zero then false)
||	logical OR (Both zero then false, either is non zero then true)
!	Is greater than

a	b	a&&b	a||b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Assignment Operators

- Assignment operators (=) is used to assign the result of an expression to a variable.
- Assignment operator stores a value in memory.
- C also supports shorthand assignment operators which simplify operation with assignment.

Operator	Meaning
=	Assigns value of right side to left side
+=	a += 1 is same as a = a + 1
-=	a -= 1 is same as a = a - 1
*=	a *= 1 is same as a = a * 1
/=	a /= 1 is same as a = a / 1
%=	a %= 1 is same as a = a % 1

Increment and Decrement Operators

- Increment (++) operator used to increase the value of the variable by one.
- Decrement (--) operator used to decrease the value of the variable by one.

Example

```
x=100;
x++;
```

Explanation

```
After the execution the
value of x will be 101.
```

Example

```
x=100;
x--;
```

Explanation

```
After the execution the
value of x will be 99.
```

Increment and Decrement Operators (cont...)

Operator	Description
Pre increment operator (++x)	value of x is incremented before assigning it to the variable on the left

Example

```
x=10;
p=++x;
```

Explanation

```
First increment value of
x by one then assign.
```

Output

```
x will be 11
p will be 11
```

Operator	Description
Post increment operator (x++)	value of x is incremented after assigning it to the variable on the left

Example

```
x=10;
p=x++;
```

Explanation

```
First assign value of x
then increment value.
```

Output

```
x will be 11
p will be 10
```

Conditional Operators

- A ternary operator is known as conditional operator.
- Syntax: *exp1 ? exp2 : exp3*

Working of the ? : Operator

```
exp1 is evaluated first
if exp1 is true(nonzero) then
        - exp2 is evaluated and its value becomes the value of the expression
If exp1 is false(zero) then
        - exp3 is evaluated and its value becomes the value of the expression
```

Example

```
m=2, n=3;
r=(m>n) ? m : n;
```

Explanation

```
Value of r will be 3
```

Example

```
m=2, n=3;
r=(m<n) ? m : n;
```

Explanation

```
Value of r will be 2
```

Bitwise Operators

- Bitwise operators are used to perform operation bit by bit.
- Bitwise operators may not be applied to float or double.

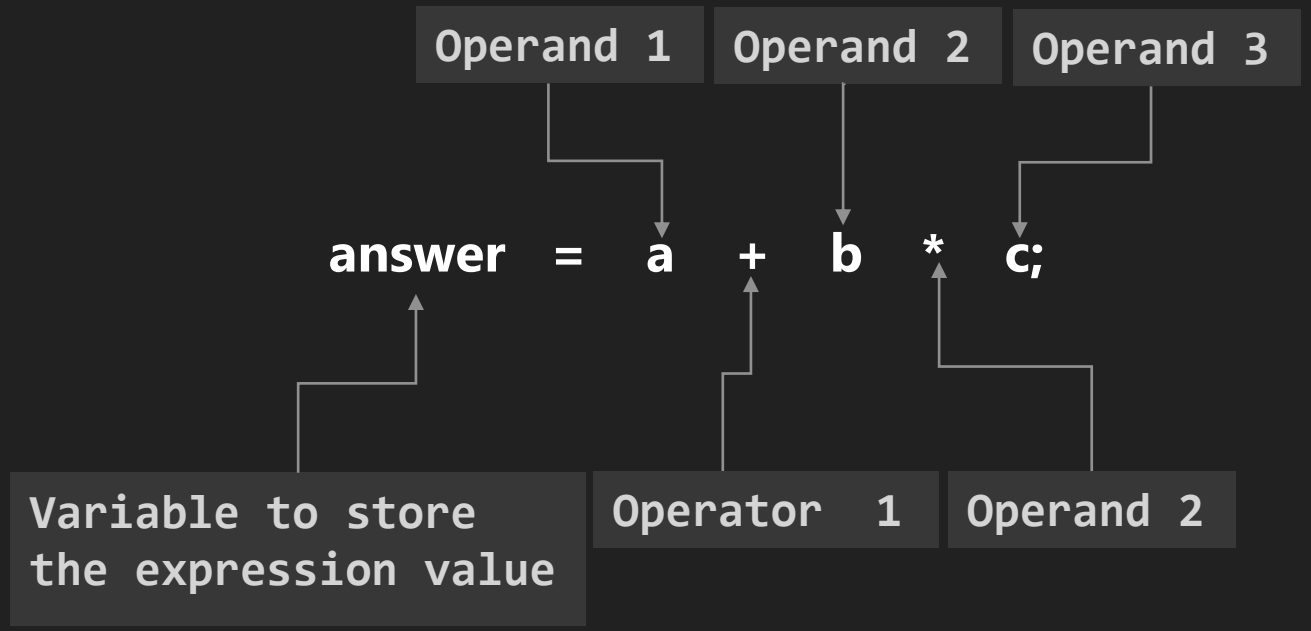
Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left (shift left means multiply by 2)
>>	shift right (shift right means divide by 2)

Special Operators

Operator	Meaning
&	Address operator, it is used to determine address of the variable.
*	Pointer operator, it is used to declare pointer variable and to get value from it.
,	Comma operator. It is used to link the related expressions together.
sizeof	It returns the number of bytes the operand occupies.
.	member selection operator, used in structure.
->	member selection operator, used in pointer to structure.

Expressions

- An expression is a combination of operators, constants and variables.
- An expression may consist of one or more operands, and zero or more operators to produce a value.



Evaluation of Expressions

- An expression is evaluated based on the operator precedence and associativity.
- When there are multiple operators in an expression, they are evaluated according to their precedence and associativity.

Operator precedence

- Precedence of an operator is its priority in an expression for evaluation.
- The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.
- Operator precedence is why the expression 5 + 3 * 2 is calculated as 5 + (3 * 2), giving 11, and not as (5 + 3) * 2, giving 16.
- We say that the multiplication operator (*) has higher "precedence" or "priority" than the addition operator (+), so the multiplication must be performed first.

Operator associativity

- Associativity is the left-to-right or right-to-left order for grouping operands to operators that have the same precedence.
- Operator associativity is why the expression 8 - 3 - 2 is calculated as (8 - 3) - 2, giving 3, and not as 8 - (3 - 2), giving 7.
- We say that the subtraction operator (-) is "left associative", so the left subtraction must be performed first.
- When we can't decide by operator precedence alone in which order to calculate an expression, we must use associativity.

Type conversion

- Type conversion is converting one type of data to another type.
- It is also known as Type Casting.
- There are two types of type conversion:
  - Implicit Type Conversion
    - This type of conversion is usually performed by the compiler when necessary without any commands by the user.
    - It is also called Automatic Type Conversion.
  - Explicit Type Conversion
    - These conversions are done explicitly by users using the pre-defined functions.

Example: Implicit Type Conversion

```
int a = 20;
double b = 20.5;
printf("%lf", a + b);
```

Output

```
40.500000
```

Example: Explicit Type Conversion

```
double a = 4.5, b = 4.6, c = 4.9;
int result = (int)da + (int)db + (int)dc;
printf("result = %d", result);
```

Output

```
12
```

printf()

- printf() is a function defined in stdio.h file
- It displays output on standard output, mostly monitor
- Message and value of variable can be printed
- Let's see few examples of printf
  - printf(" ");
  - printf("Hello World"); // Hello World
  - printf("%d", c); // 15
  - printf("Sum = %d", c); // Sum = 15
  - printf("%d+%d=%d", a, b, c); // 10+5=15

scanf()

- scanf() is a function defined in stdio.h file
- scanf() function is used to read character, string, numeric data from keyboard
- Syntax of scanf
  - `scanf("%X", &variable);`
    - where %X is the format specifier which tells the compiler what type of data is in a variable.
    - & refers to address of “variable” which is directing the input value to a address returned by &variable.

Format specifier	Supported data types	Example	Description
%d	Integer	scanf(“%d”, &a)	Accept integer value such as 1, 5, 25, 105 etc
%f	Float	scanf(“%f”, &b)	Accept floating value such as 1.5, 15.20 etc
%c	Character	scanf(“%c”, &c)	Accept character value such as a, f, j, W, Z etc
%s	String	scanf(“%s”, &d)	Accept string value such as diet, india etc

getchar and putchar

- getchar function reads a single character from terminal.
- putchar function displays the character passed to it on the screen.

Program

```c
#include <stdio.h>
void main( )
{
    int c;
    printf("Enter a character: ");
    /* Take a character as input */
    c = getchar();
    /* Display the character */
    printf("Entered character is: ");
    putchar(c);
}
```

Output

```
Enter a character: a
Entered character is: a
```

gets and puts

- gets function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF (End of File) occurs.
- puts function writes the string 's' and 'a' trailing newline to stdout.

Program

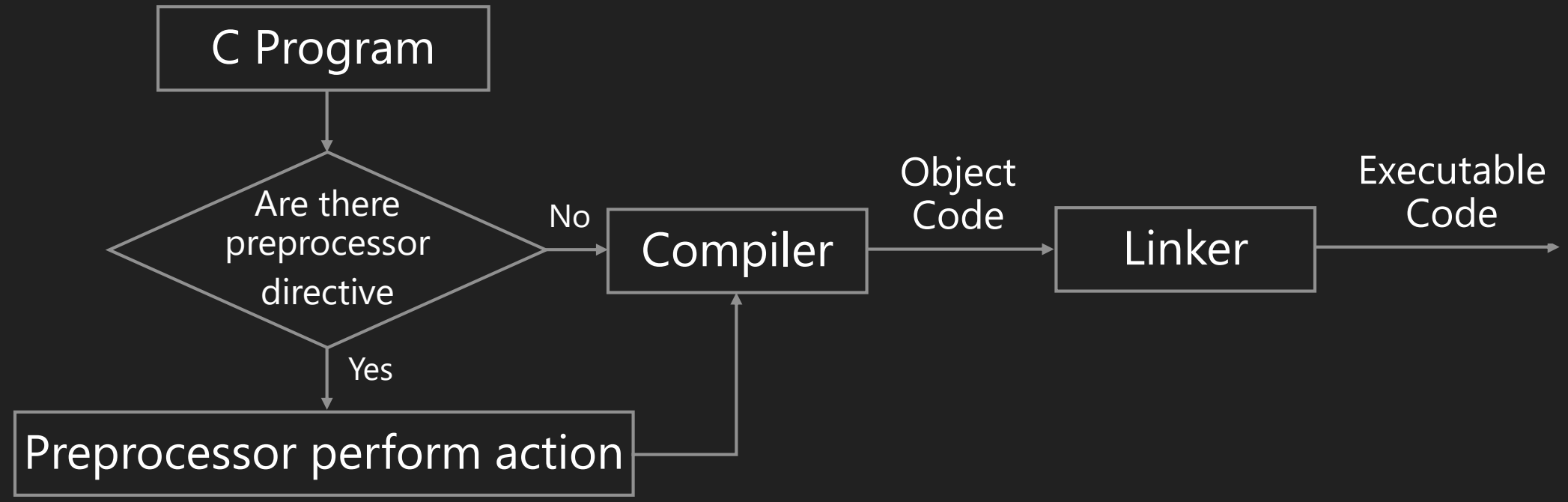
```c
#include <stdio.h>
void main( )
{
    /*Character array of length 100*/
    char str[100];
    printf("Enter a string: ");
    /* Take a string as input */
    gets( str );
    /* Display the string */
    printf("Entered string is: ");
    puts( str );
}
```

Output

```
Enter a string: india
Entered string is: india
```

Preprocessor

- Preprocessors are programs that process our source code before compilation.
- There are a number of steps involved between writing a program and executing a program in C.
- Let us have a look at these steps before we actually start learning about Preprocessors.



Types of Preprocessor

- There are 4 main types of preprocessor directives:
  - Macros
  - File inclusion
  - Conditional compilation
  - Other directives

Macro

- A macro is a fragment of code which has been given a name. Whenever the name is used in program, it is replaced by the contents of the macro.
- Macro definitions are not variables and cannot be changed by your program code like variables.
- The ‘#define’ directive is used to define a macro.
- Do not put a semicolon (;) at the end of #define statements.
- There are two types of macros:
  - Object-like Macros
  - Function-like Macros

Macro

Description	Object-like Macros	Function-like Macros
Definition	The object-like macro is an identifier that is replaced by value.	The function-like macro looks like function call.
Use	It is used to represent numeric constants.	It is used to represent function.
Syntax	#define CNAME value	#define CNAME (expression)
Example	#define PI 3.14	#define MIN(a,b) ((a)<(b)?(a):(b))
Program		

```c
#include <stdio.h>
#define PI 3.14
void main()
{   int r=2;
    float a;
    a=PI*r*r;
    printf("%f", a);
}
```

```c
#include <stdio.h>
#define MIN(a,b) ((a)<(b)?(a):(b))
void main()
{
    printf("%d", MIN(2, 5));
}
```

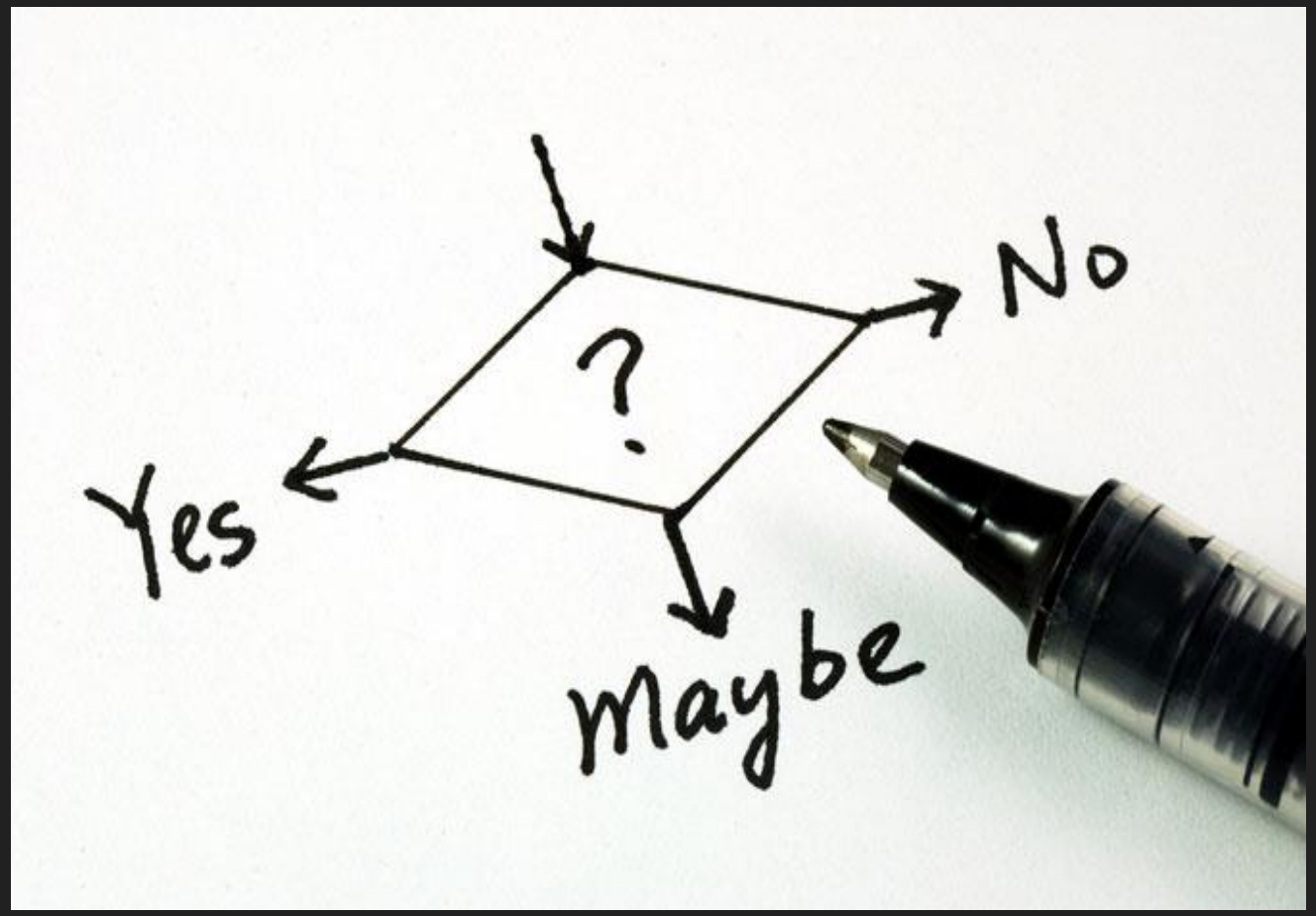
Decision making in C

USING

{C}

Programming

Need of decision making




```
if number is odd
{
    /* code */
}
```

```
else number is even
{
    /* code */
}
```

Decision Making or Conditional Statement

- C program statements are executed sequentially.
- Decision Making statements are used to control the flow of program.
- It allows us to control whether a program segment is executed or not.
- It evaluates condition or logical expression first and based on its result (either true or false), the control is transferred to particular statement.
- If result is true then it takes one path else it takes another path.

Decision Making Statements in C

Decision Making Statements are

One way Decision:	`if` (Also known as simple if)
Two way Decision:	`if…else`
Multi way Decision:	`if…else if…else if…else`
Two way Decision:	`?:` (Conditional Operator)
n-way Decision:	`switch…case`

Relational Operators

- Relational Operator is used to compare two expressions.
- It gives result either true or false based on relationship of two expressions.

Math	C	Meaning	Example	Result
>	>	is greater than	`5 > 4`	`true`
≥	>=	is greater than or equal to	`5 >= 4`	`true`
<	<	is less than	`5 < 4`	`false`
≤	<=	is less than or equal to	`5 <= 4`	`false`
≠	!=	is not equal to	`5 != 4`	`true`
=	==	is equal to	`5 == 4`	`false`

If statement

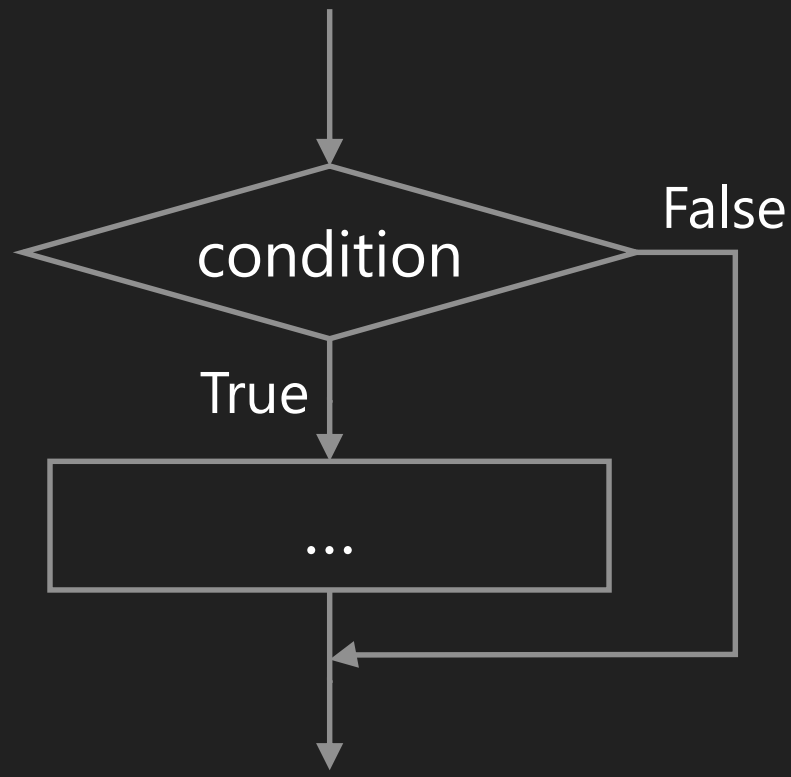
if

- `if` is single branch decision making statement.
- If condition is `true` then only body will be executed.
- `if` is a keyword.

Syntax

```
if(condition)
{
    // Body of the if
    // true part
}
```

Flowchart of `if`



WAP to print Zero if given number is 0

Program

```c
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a == 0)
    {
        printf("Zero");
    }
}
```

Output

```
Enter Number:0
Zero
```

WAP to print Positive or Negative Number

Program

```c
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a >= 0)
    {
        printf("Positive Number");
    }
    if(a < 0)
    {
        printf("Negative Number");
    }
}
```

Output

```
Enter Number:5
Positive Number
```

Output

```
Enter Number:-5
Negative Number
```

Modulus Operator

- `%` is modulus operator in C
- It divides the value of one expression (number) by the value of another expression (number), and returns the remainder.
- Syntax: `express1 % express2`
- E.g.
  - `7%2` Answer: 1
  - `6%2` Answer: 0
  - `25%10` Answer: 5
  - `37%28` Answer: 9

WAP to print Odd or Even Number

Program

```c
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a%2 == 0)
    {
        printf("Even Number");
    }
    if(a%2 != 0)
    {
        printf("Odd Number");
    }
}
```

Output

```
Enter Number:12
Even Number
```

Output

```
Enter Number:11
Odd Number
```

If..else statement

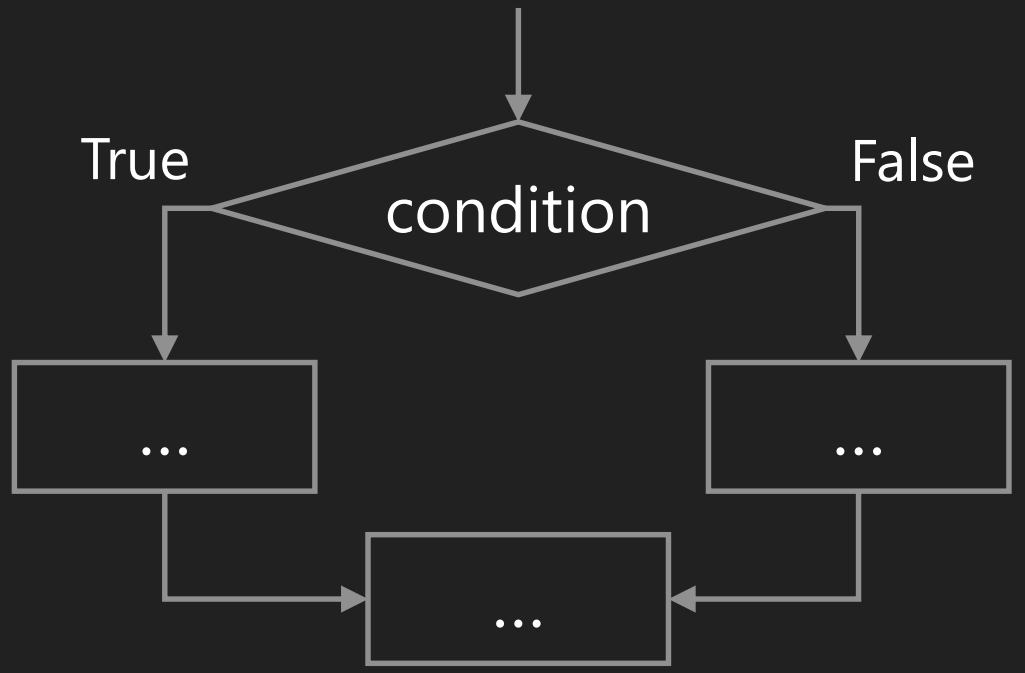
if...else

- `if…else` is two branch decision making statement
- If condition is true then true part will be executed else false part will be executed
- `else` is keyword

Syntax

```
if(condition)
{
    // true part
}
else
{
    // false part
}
```

Flowchart of `if…else`



WAP to print Positive or Negative Number using if...else

Program

```c
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a >= 0)
    {
        printf("Positive Number");
    }
    else
    {
        printf("Negative Number");
    }
}
```

Output

```
Enter Number:5
Positive Number
```

Output

```
Enter Number:-5
Negative Number
```

WAP to print Odd or Even Number using if...else

Program

```c
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a%2 == 0)
    {
        printf("Even Number");
    }
    else
    {
        printf("Odd Number");
    }
}
```

Output

```
Enter Number:12
Even Number
```

Output

```
Enter Number:11
Odd Number
```

WAP to find largest number from given 2 numbers using if

Program

```c
#include<stdio.h>
void main()
{
    int a, b;
    printf("Enter Two Numbers:");
    scanf("%d%d",&a,&b);
    if(a > b)
    {
        printf("%d is largest", a);
    }
    if(a < b)
    {
         printf("%d is largest", b);
    }
}
```

Output

```
Enter Two Numbers:4
5
5 is largest
```

WAP to find largest number from given 2 numbers using if...else

Program

```c
#include<stdio.h>
void main()
{
    int a, b;
    printf("Enter Two Numbers:");
    scanf("%d%d",&a,&b);
    if(a > b)
    {
        printf("%d is largest", a);
    }
    else
    {
         printf("%d is largest", b);
    }
}
```

Output

```
Enter Two Numbers:4
5
5 is largest
```

{ }

- If body of `if` contains only one statement then `{ }` are not compulsory
- But if body of `if` contains more than one statements then `{ }` are compulsory

```
if(a >= b)
{
    printf("%d is largest", a);
}
```

Both are same

```
if(a >= b)
    printf("%d is largest", a);
```

If...else if...else if...else Ladder if

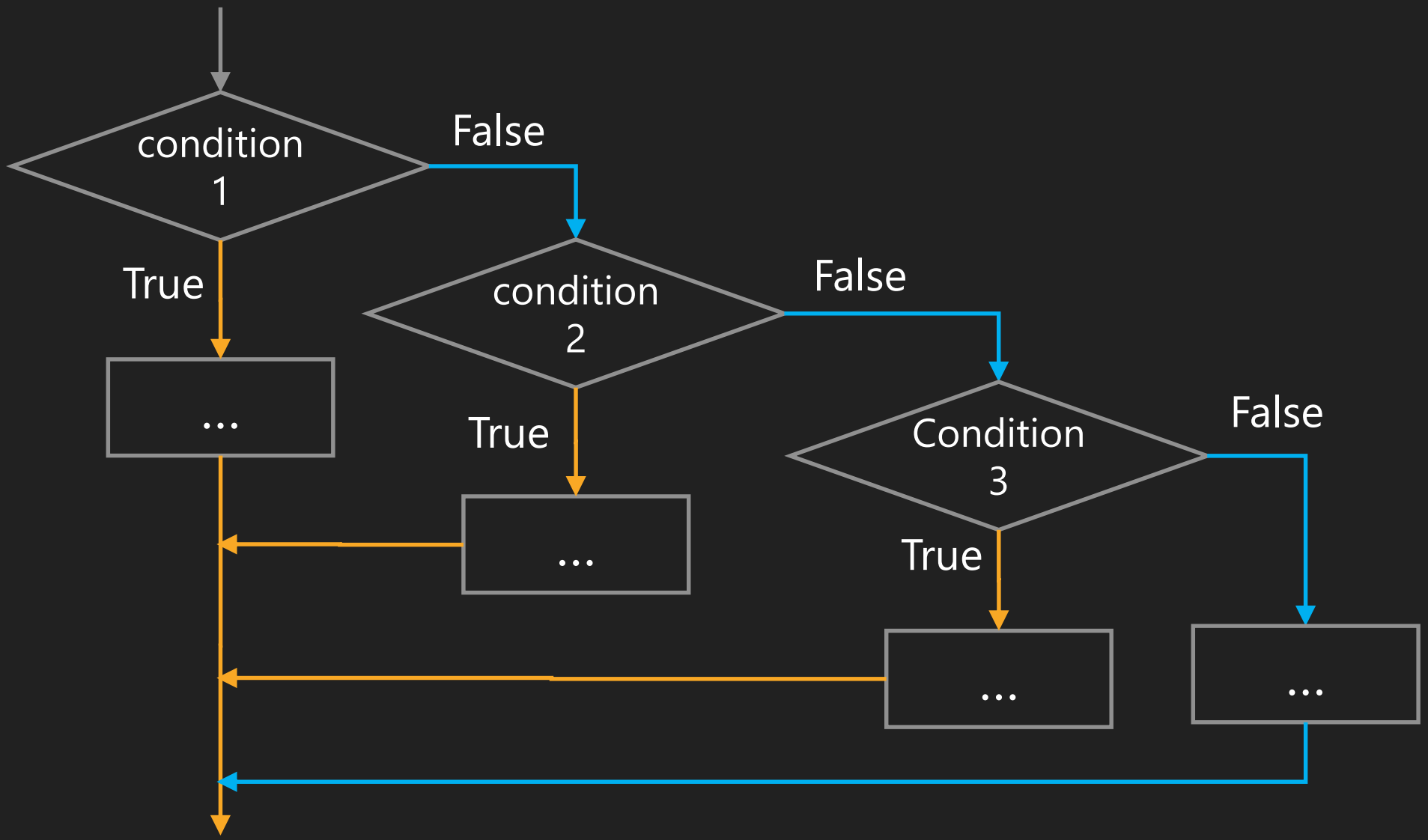
If…else if…else if…else

- **if…else if…else if…else** is multi branch decision making statement.
- If first **if** condition is true then remaining **if** conditions will not be evaluated.
- If first **if** condition is false then second **if** condition will be evaluated and if it is true then remaining **if** conditions will not be evaluated.
- **if…else if…else if…else** is also known as if…else if ladder

Syntax

```
if(condition-1)
  statement-1;
else if(condition-2)
  statement-2;
else
  statement-3;
```

`if…else if…else` ladder flowchart



WAP to print Zero, Positive or Negative Number

Program

```c
#include<stdio.h>
void main()
{
    int a;
    printf("Enter Number:");
    scanf("%d",&a);
    if(a > 0)
        printf("Positive Number");
    else if(a==0)
        printf("Zero");
    else
        printf("Negative Number");
}
```

Output

```
Enter Number:5
Positive Number
```

Output

```
Enter Number:-5
Negative Number
```

Nested if

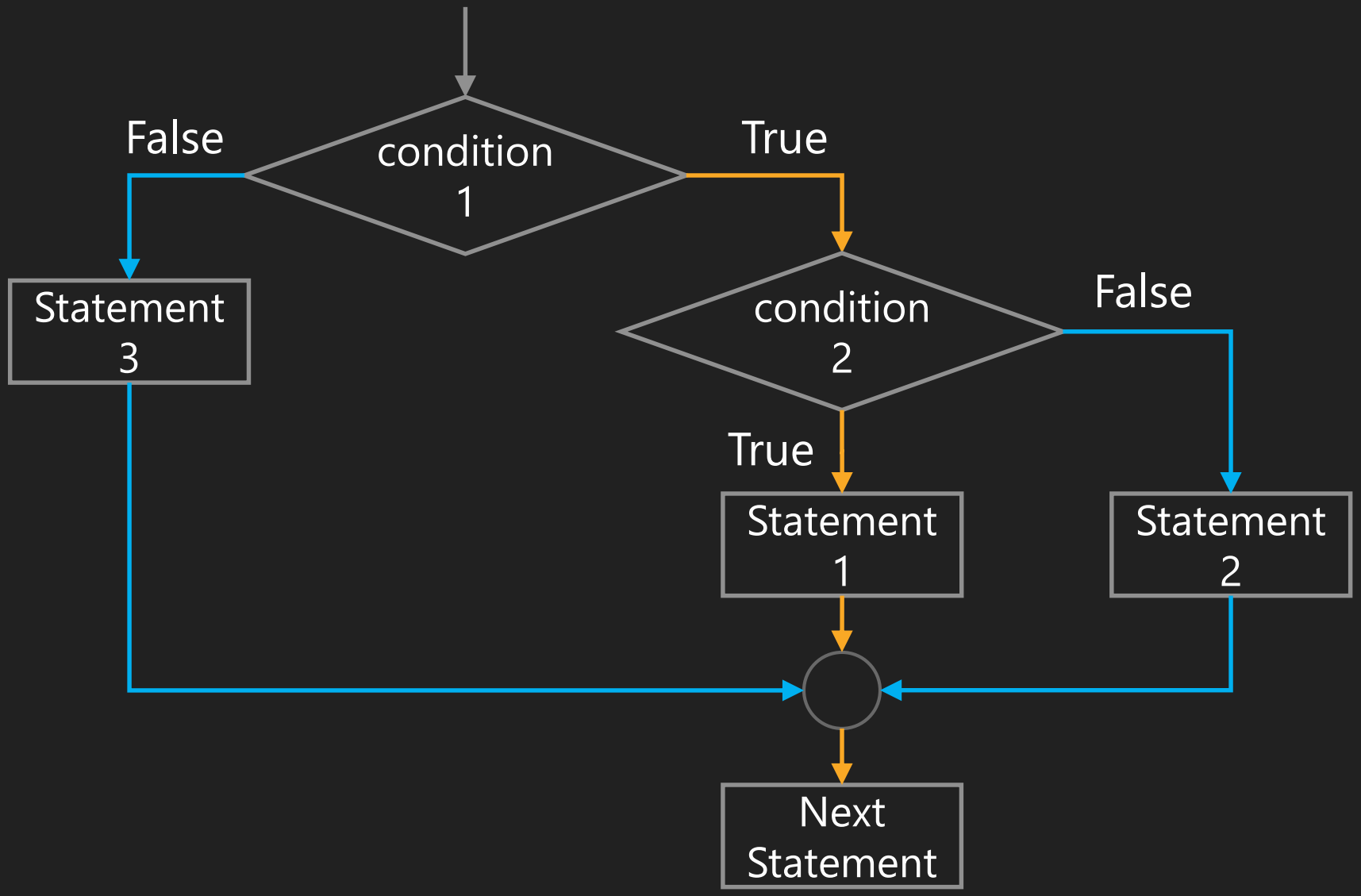
Nested if

- If condition-1 is true then condition-2 is evaluated. If it is true then statement-1 will be executed.
- If condition-1 is false then statement-3 will be executed.

Syntax

```
if(condition-1)
{
    if(condition-2)
    {
      statement-1;
    }
    else
    {
      statement-2;
    }
}
else
{
  statement-3;
}
```

Nested `if` flowchart



WAP to print maximum from given three numbers

Program

```c
void main(){
    int a, b, c;
    printf("Enter Three Numbers:");
    scanf("%d%d%d",&a,&b,&c);
    if(a>b)
    {
      if(a>c)
        printf("%d is max",a);
      else
        printf("%d is max",c);
   }
   else
   {
     if(b>c)
       printf("%d is max",b);
     else
       printf("%d is max",c);
   }
}
```

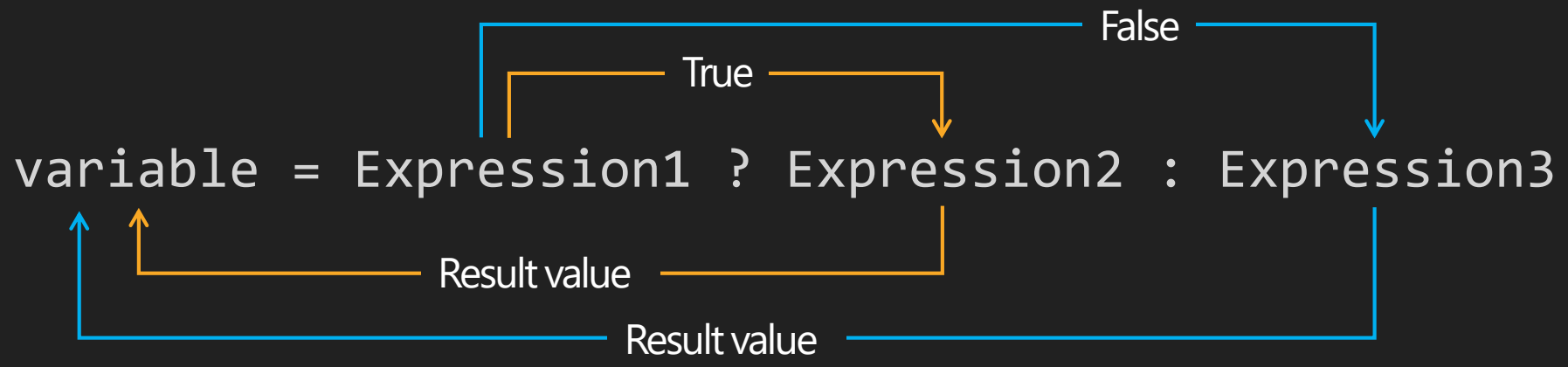
Output

```
Enter Three Numbers:7
5
9
9 is max
```

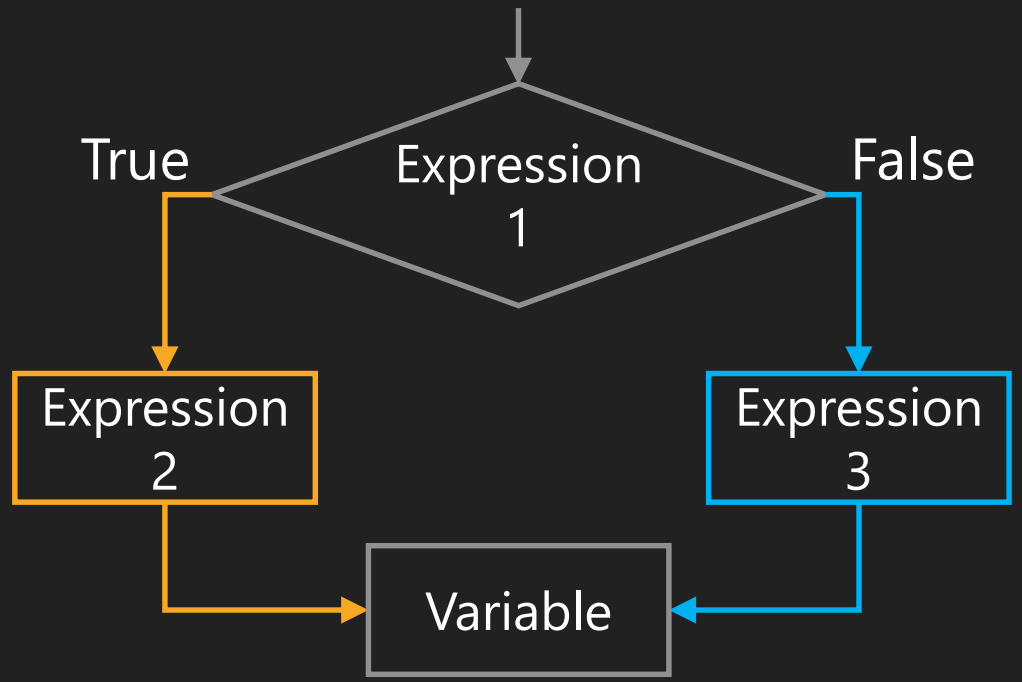
Conditional Operator

? : (Conditional Operator)

- The conditional works operator is similar to the if-else.
- It is also known as a ternary operator.
- It returns first value of expression (before colon(:)) if expression is true and second value of expression if expression is false.



Conditional operator flowchart



- Here, Expression1 is the condition to be evaluated.
- If the condition(Expression1) is True then Expression2 will be executed and the result will be returned.
- Otherwise, if condition(Expression1) is false then Expression3 will be executed and the result will be returned.

WAP to find largest number from given 2 numbers using ? :

Program

```c
#include<stdio.h>
void main()
{
    int a, b, max;
    printf("Enter Two Numbers:");
    scanf("%d%d",&a,&b);
    max = a>b?a:b;
    printf("%d is largest",max);
}
```

Output

```
Enter Two Numbers:4
5
5 is largest
```

switch...case

switch...case

- The switch statement allows to execute one code block among many alternatives.
- It works similar to if...else..if ladder.

Syntax

```
switch (expression)
{
    case constant1:
      // statements
      break;
    case constant2:
      // statements
      break;
    .
    .
    .
    default:
      // default statements
}
```

- The expression is evaluated once and compared with the values of each `case`.
- If there is a match, the corresponding statements after the matching `case` are executed.
- If there is no match, the `default` statements are executed.
- If we do not use `break`, all statements after the matching label are executed.
- The `default` clause inside the `switch` statement is optional.

WAP that asks day number and prints day name using switch…case

```c
void main(){
    int day;
    printf("Enter day number(1-7):");
    scanf("%d",&day);
    switch(day)
    {
        case 1:
                printf("Sunday");
                break;
        case 2:
                printf("Monday");
                break;
        case 3:
                printf("Tuesday");
                break;
        case 4:
                printf("Wednesday");
                break;
        case 5:
                printf("Thursday");
                break;
        case 6:
                printf("Friday");
                break;
        case 7:
                printf("Saturday");
                break;
        default:
                printf("Wrong input");
                break;
    }
}
```

Output

```
Enter day number(1-7):5
Thursday
```

Practice programs

1) Write a program to check whether entered character is vowel or not?
2) Write a program to perform Addition, Subtraction, Multiplication and Division of 2 numbers as per user's choice (using if…else/Nested if/Ladder if).
3) Write a program to read marks of five subjects. Calculate percentage and print class accordingly. Fail below 35, Pass Class between 35 to 45, Second Class between 45 to 60, First Class between 60 to 70, Distinction if more than 70.
4) Write a program to find out largest number from given 3 numbers (Conditional operator).
5) Write a program to print number of days in the given month.

Thank you

Looping

USING

{C}

Programming

Life is all about Repetition.

We do same thing everyday

What is loop?

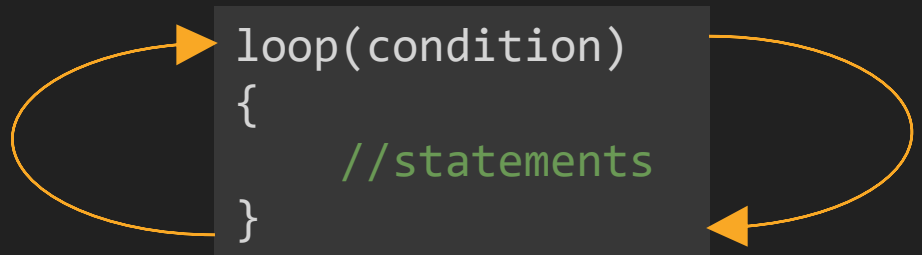
- Loop is used to execute the block of code several times according to the condition given in the loop. It means it executes the same code multiple times.

"Hello" 5

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

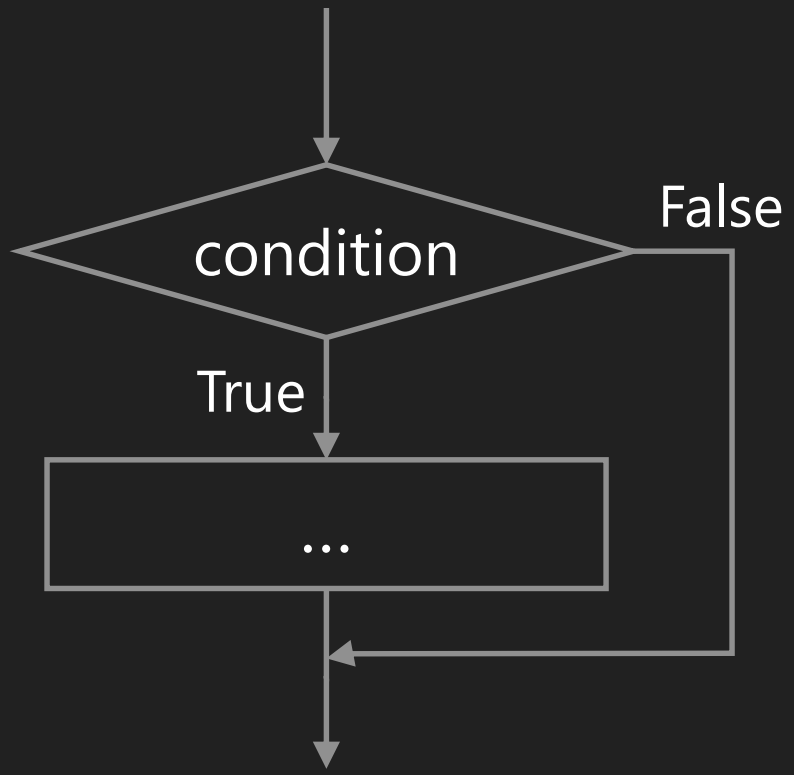
Output

```
Hello
Hello
Hello
Hello
Hello
```



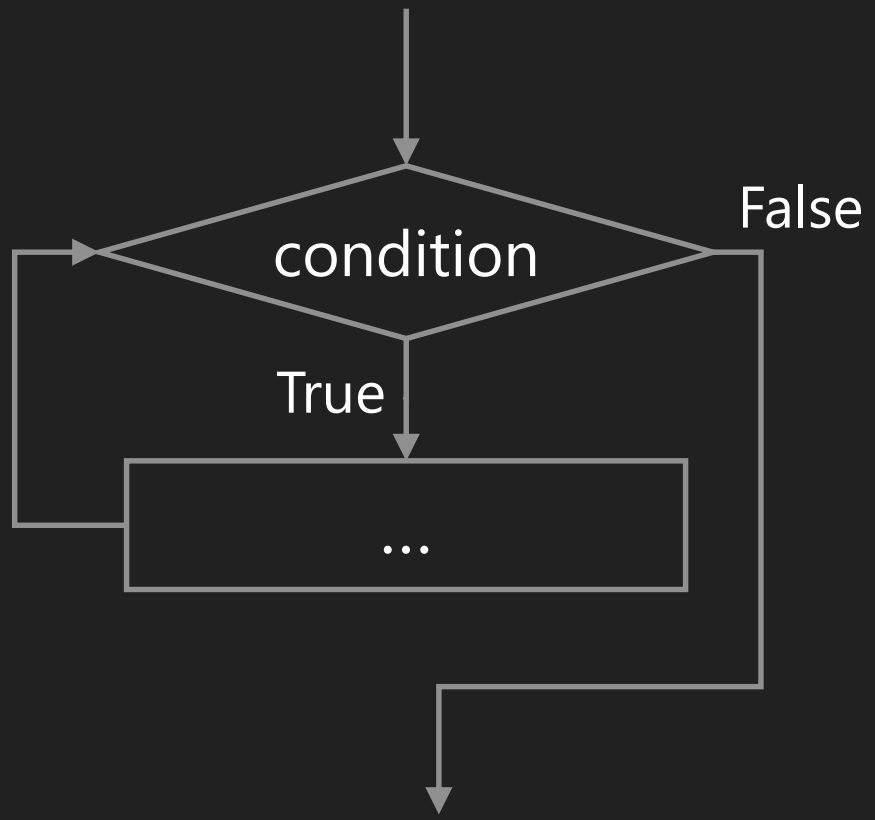
`if` v/s `while`

Flowchart of `if`



v/s

Flowchart of `while`



Looping or Iterative Statements in C

Looping Statements are

Entry Controlled Loop:	`while`, `for`
Exit Controlled Loop:	`do…while`
Virtual Loop:	`goto`

While loop

`While` Loop

- `while` is an entry controlled loop
- Statements inside the body of `while` are repeatedly executed till the condition is true
- `while` is keyword

Syntax

```
while(condition)
{
    // Body of the while
    // true part
}
```

WAP to print 1 to n(while loop)

Program

```c
#include <stdio.h>
void main()
{
    int i,n;
    i=1;
    printf("Enter n:");
    scanf("%d",&n);
    while(i<=n)
    {
        printf("%d\n",i);
        i=i+1;
    }
}
```

Output

```
Enter n:10
1
2
3
4
5
6
7
8
9
10
```

WAP to print multiplication table(while loop)

Program

```c
#include<stdio.h>
void main()
{
    int i=1,n;
    printf("Enter n for multiplication table:");
    scanf("%d",&n);
    while(i<=10)
    {
        printf("%d * %d = %d\n",n,i,n*i);
        i=i+1;
    }
}
```

Output

```
Enter n for multiplication table:5
5 * 1  = 5
5 * 2  = 10
5 * 3  = 15
5 * 4  = 20
5 * 5  = 25
5 * 6  = 30
5 * 7  = 35
5 * 8  = 40
5 * 9  = 45
5 * 10 = 50
```

WAP to Sum of 5 numbers entered by user(while loop)

Program

```c
#include<stdio.h>
void main()
{
    int sum=0, i=1,n;
    while(i<=5)
    {
        printf("Enter a number=");
        scanf("%d",&n);
        sum=sum+n;
        i=i+1;
    }
    printf("Sum is=%d",sum);
}
```

Output

```
Enter a number=10
Enter a number=20
Enter a number=30
Enter a number=40
Enter a number=50
Sum is=150
```

Syntax and Logic

Swimming Rules

1. Breath control
2. Kicking legs
3. Back stroke with arms
4. Front stroke with arms
5. Crawling in water

To Swim

Syntax

```
while(condition)
{
    // Body of the while
    // true part
}
```

Logic

```
int i = 1;
while (i <= 5)
{
    printf("%d\n", i);
    i=i+1;
}
```

How to build logic? Step-1

Step 1: Understand the problem statement

- e.g. Write a program to find factors of a number.
- Run following questions through mind

- What is the factor of a number?
  - Factor is a number that divides another number evenly with no remainder.
  - For example, 1,2,3,4,6,12 are factors of 12.
- How many variables needed? What should be their data types?(Inputs/Outputs)
  - To get number from user we need variable **`n`**.
  - Now we need to divide **`n`** with 1,2,3,...,n. For this we will declare a loop variable **`i`** initialized as 1.
  - Both variables should be of `integer` data type.
- What control structure you require?
  - First we need `a loop` to divide **`n`** by 1,2,3,...,n, loop will start from 1 and ends at **`n`**.
  - Inside loop we need `if structure` to check **`n%i==0`** (Number n is evenly divisible by **`i`** or not).

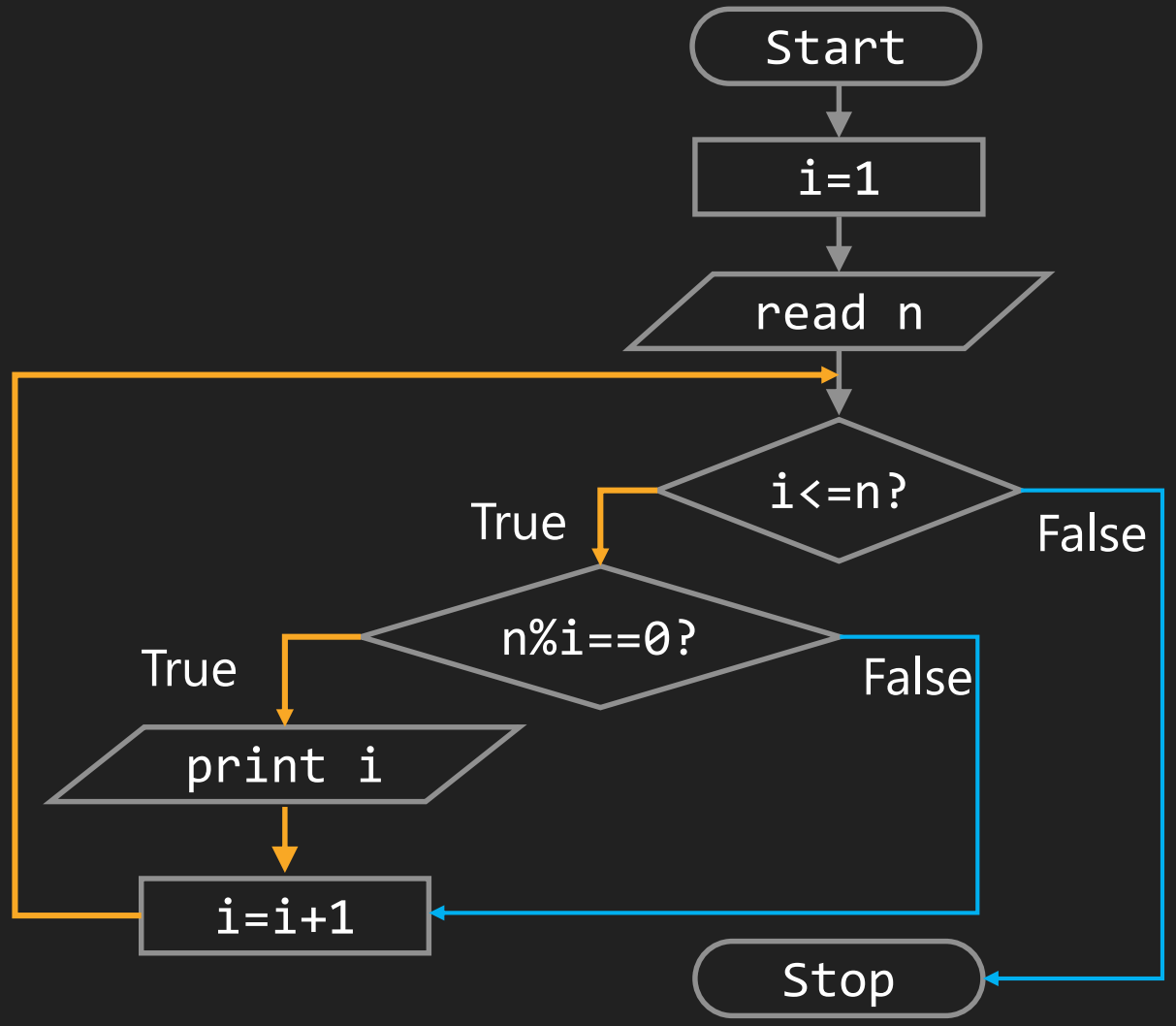
How to build logic? Step-2

Step 2: Think for 1 or 2 examples

- Consider n=6, now take `i=1`
  - `6%1==0`, TRUE; So, 1 is factor of 6
  - `6%2==0`, TRUE; So, 2 is factor of 6
  - `6%3==0`, TRUE; So, 3 is factor of 6
  - `6%4==2`, FALSE; S0, 4 is not factor of 6
  - `6%5==1`, FALSE; S0, 5 is not factor of 6
  - `6%6==0`, TRUE; S0, 6 is factor of 6
- From this we can infer that loop variable **`i`** starts with `1` and incremented by one for next iteration then ends at value **`n`**.
- Consider n=`10`, factors are `1,2,5,10`
- Consider n=`11`, factor is `1,11`
- From this we can infer that 1 and number itself are always factors of any number **`n`**.

How to build logic? Step-3

Step 3: Draw flowchart/steps on paper or in mind



Steps

```
Step 1: Start
Step 2: Declare variables n,i
Step 3: Initialize variable
        i ← 1
Step 4: Read value of n
Step 5: Repeat the steps until i = n
     5.1: if n%i == 0
             Display i
     5.2: i=i+1
Step 7: Stop
```

How to build logic? Step-4

Step 4: Writing Pseudo-code

- Pseudo-code is an informal way to express the design of a computer program or an algorithm.
- It does not require any strict programming language syntax.

Pseudo-code

```
Initialize i=1 integer
Declare n as integer
Input n
while i<n
    if n%i
        print i
    end if
    increment i=i+1
end while
```

WAP to find factors of a number(while loop)

Program

```c
#include <stdio.h>
void main()
{
    int i=1,n;
    printf("Enter n to find factors=");
    scanf("%d",&n);
    while(i<=n)
    {
        if(n%i==0)
            printf("%d,",i);
        i=i+1;
    }
}
```

Output

```
Enter n to find factors=12
1,2,3,4,6,12,
```

WAP to print reverse a number(while loop)

Program

```c
#include <stdio.h>
void main()
{
    int n;
    printf("Enter a number=");
    scanf("%d",&n);
    while(n!=0)
    {
        printf("%d",n%10);
        n=n/10;
    }
}
```

Output

```
Enter a number=1234
4321
```

WAP to check given number is perfect or not(while loop)

```c
void main(){
    int i=1,n,sum=0;
    printf("Enter a number:");
    scanf("%d",&n);
    while(i<n)
    {
        if(n%i==0)
        {
            printf("%d+",i);
            sum=sum+i;
        }
        i=i+1;
    }
    printf("=%d",sum);
    if(sum==n)
        printf("\n%d is a perfect number",n);
    else
        printf("\n%d is not a perfect number",n);
}
```

Output

```
Enter a number:6
1+2+3=6
6 is a perfect number
```

Output

```
Enter a number:8
1+2+4+=7
8 is not a perfect number
```

Output

```
Enter a number:496
1+2+4+8+16+31+62+124+248+=496
496 is a perfect number
```

WAP to check given number is prime or not(while loop)

```c
void main()
{
    int n, i=2,flag=0;
    printf("Enter a number:");
    scanf("%d",&n);
    while(i<=n/2)
    {
        if(n%i==0)
        {
            flag=1;
            break;
        }
        i++;
    }
    if (flag==0)
        printf("%d is a prime number",n);
    else
        printf("%d is not a prime number",n);
}
```

Output

```
Enter a number:7
7 is a prime number
```

Output

```
Enter a number:9
9 is not a prime number
```

for loop

`for` Loop

- `for` is an entry controlled loop
- Statements inside the body of `for` are repeatedly executed till the condition is true
- `for` is keyword

Syntax

```
for (initialization; condition; updateStatement)
{
    // statements
}
```

- The initialization statement is executed only once.
- Then, the condition is evaluated. If the condition is `false`, the `for` loop is terminated.
- If the condition is `true`, statements inside the body of for loop are executed, and the update statement is updated.
- Again the condition is evaluated.

WAP to print numbers 1 to n (for loop)

Program

```c
#include<stdio.h>
void main()
{
    int i,n;
    printf("Enter a number:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("%d\n",i);
    }
}
```

Output

```
Enter a number:5
1
2
3
4
5
```

WAP to find factors of a number (for loop)

Program

```c
#include <stdio.h>
void main()
{
    int i,n;
    printf("Enter n to find factors=");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        if(n%i==0)
            printf("%d,",i);
    }
}
```

Output

```
Enter n to find factors=12
1,2,3,4,6,12,
```

WAP to check given number is perfect or not(for loop)

```c
void main(){
    int i,n,sum=0;
    printf("Enter a number:");
    scanf("%d",&n);
    for(i=1;i<n;i++)
    {
        if(n%i==0)
        {
            printf("%d+",i);
            sum=sum+i;
        }
    }
    printf("=%d",sum);
    if(sum==n)
        printf("\n%d is a perfect number",n);
    else
        printf("\n%d is not a perfect number",n);
}
```

Output

```
Enter a number:6
1+2+3=6
6 is a perfect number
```

Output

```
Enter a number:8
1+2+4+=7
8 is not a perfect number
```

Output

```
Enter a number:496
1+2+4+8+16+31+62+124+248+=496
496 is a perfect number
```

do while loop

`do while` Loop

- `do while` is an exit controlled loop.
- Statements inside the body of `do while` are repeatedly executed till the condition is true.
- `Do` and `while` are keywords.

Syntax

```
do
{
    // statement
}
while (condition);
```

- Loop body will be executed first, and then condition is checked.
- If the condition is true, the body of the loop is executed again and the condition is evaluated.
- This process goes on until the condition becomes false.
- If the condition is false, the loop ends.

WAP to print Odd numbers between 1 to n(do while loop)

Program

```c
void main()
{
   int i=1,n;
   printf("Enter a number:");
   scanf("%d",&n);
   do
   {
      if(i%2!=0)
      {
         printf("%d,",i);
      }
      i=i+1;
   }
   while(i<=n);
}
```

Output

```
Enter a number:5
1,3,5
```

WAP to find factors of a number(do while loop)

Program

```
1  void main()
2  {
3     int i=1,n;
4     printf("Enter a number:");
5     scanf("%d",&n);
6     do
7     {
8        if(n%i==0)
9        {
10          printf("%d,",i);
11       }
12       i=i+1;
13    }
14    while(i<=n);
15 }
```

Output

```
Enter a number:6
1,2,3,6,
```

WAP to print reverse a number(do while loop)

Program

```c
void main()
{
    int n;
    printf("Enter a number:");
    scanf("%d",&n);
    do
    {
        printf("%d",n%10);
        n=n/10;
    }
    while(n!=0);
}
```

Output

```
Enter a number=1234
4321
```

goto statement

`goto` Statement

- `goto` is an virtual loop
- The `goto` statement allows us to transfer control of the program to the specified `label`.
- `goto` is keyword

Syntax

```
goto label;
.
.
.
label:
```

Syntax

```
label:
.
.
.
goto label;
```

- The `label` is an identifier. When the `goto` statement is encountered, the control of the program jumps to `label:` and starts executing the code.

WAP to print Odd numbers between 1 to n(goto)

Program

```c
void main()
{
    int i=1,n;
    printf("Enter a number:");
    scanf("%d",&n);
    odd:
    if(i%2!=0)
    {
        printf("%d,",i);
    }
    i=i+1;
    if(i<=n)
    {
        goto odd;
    }
}
```

Output

```
Enter a number:5
1,3,5
```

WAP to find factors of a number(goto)

Program

```c
void main()
{
    int i=1,n;
    printf("Enter a number:");
    scanf("%d",&n);
    odd:
    if(n%i==0)
    {
        printf("%d,",i);
    }
    i=i+1;
    if(i<=n)
    {
        goto odd;
    }
}
```

Output

```
Enter a number:6
1,2,3,6,
```

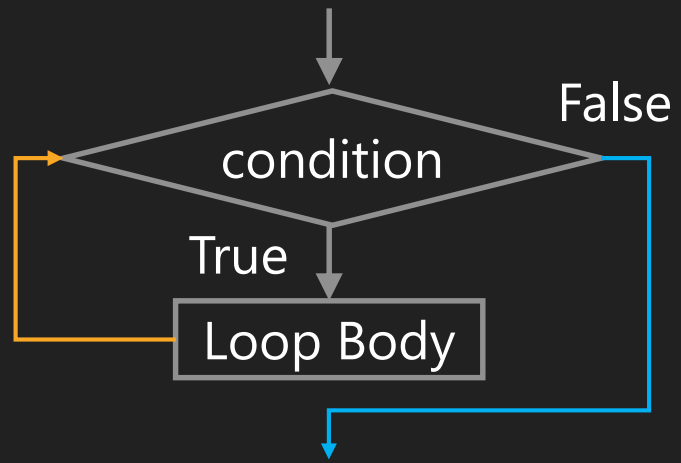
Types of loops

Entry Control Loop

```
int i=1;
while(i<=10)
{
    printf("%d",i++);
}
```

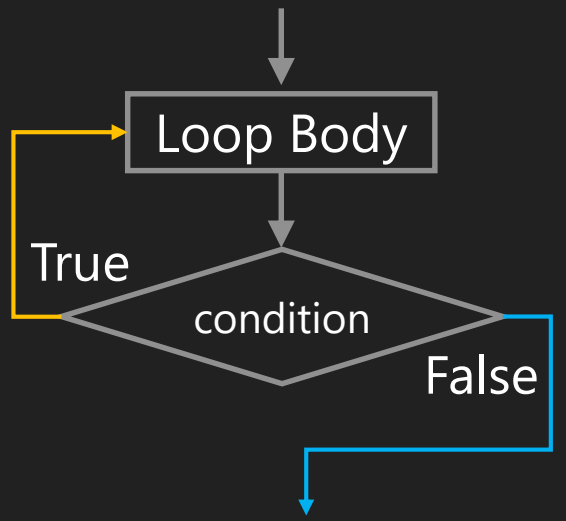
Entry Control Loop

```
int i;
for(i=1;i<=10;i++)
{
    printf("%d",i);
}
```



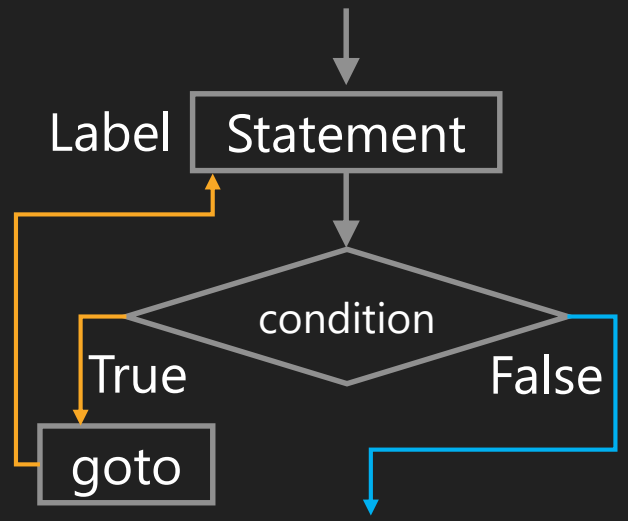
Exit Control Loop

```
int i=1;
do
{
    printf("%d",i++);
}
while(i<=10);
```



Virtual Loop

```
int i=1;
   labelprint:
    printf("%d",i++);
   if(i<=10)
     goto labelprint;
```



Pattern

Always detect pattern in pattern

Pattern

There are important points to note in pattern

1. Determine, how many rows?
2. Determine, how many numbers/characters/columns in a row?
3. Determine, Increment/Decrement among the number of rows.
4. Determine, starting in each row

```
1
11
111
1111
11111
```

```
1
12
123
1234
12345
```

```
1
23
456
78910
```

```
    *
   * *
  * * *
 * * * *
  * * *
   * *
    *
```

WAP to print given pattern (nested loop)

```
*
**
***
****
*****
```

No. of rows: 5

No. of characters
Row-1: *
Row-2: **
Row-3: ***
Row-4: ****
Row-5: *****

Inner loop: Increment
Outer loop: Increment

Starting: *

Program

```c
void main()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        for(j=1; j<=i; j++)
        {
            printf("*");
        }
        printf("\n");
    }
}
```

WAP to print given pattern (nested loop)

```
1
12
123
1234
12345
```

No. of rows: 5

No. of values
Row-1: 1
Row-2: 12
Row-3: 123
Row-4: 1234
Row-5: 12345

Inner loop: Increment
Outer loop: Increment

Starting: 1

Program

```c
void main()
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        for(j=1; j<=i; j++)
        {
            printf("%d",j);
        }
        printf("\n");
    }
}
```

WAP to print given pattern (nested loop)

```
5
54
543
5432
54321
```

No. of rows: 5

No. of values
Row-1: 5
Row-2: 54
Row-3: 543
Row-4: 5432
Row-5: 54321

Inner loop: Decrement
Outer loop: Decrement/Increment

Starting: 5

Program

```c
void main()
{
    int i,j;
    for(i=5;i>0;i--)
    {
        for(j=5; j>=i ; j--)
        {
            printf("%d",j);
        }
        printf("\n");
    }
}
```

WAP to print given pattern (nested loop)

```
    *
   **
  ***
 ****
*****
```

No. of rows: 5

No. of values
Row-1: ----*
Row-2: ---**
Row-3: --***
Row-4: -****
Row-5: *****

Inner loop: Decrement
Outer loop: Decrement/Increment

Starting: -(space)
Ending: *

Program

```
1  void main()
2  {
3      int i,j,k;
4      for(i=1;i<=5;i++)
5      {
6          for(k=5;k>i;k--)
7          {
8              printf(" ");
9          }
10         for(j=1;j<=i;j++)
11         {
12             printf("*");
13         }
14         printf("\n");
15     }
16 }
```

First we need to print 4 spaces before printing *

```
    *
   **
  ***
 ****
*****
```

After printing spaces this inner loop prints *

Practice programs

1) Write a program to find sum of first N odd numbers. Ex. 1+3+5+7+.....+N
2) Write a program to find 1+1/2+1/3+1/4+....+1/n.
3) Write a program to print all Armstrong numbers in a given range. For example 153 = 1^3 + 5^3 + 3^3. So, 153 is Armstrong number.
4) Write a program to print given number in reverse order
5) Write a program to check whether a given string is palindrome or not.
6) Write a program to print Multiplication Table up to n.

1	2	3	4	5	6	7	.
2	4	6	8	10	12	14	.
3	6	9	12	15	18	21	.
4	8	12	16	20	24	28	.
5	10	15	20	25	30	35	.
.	.	.	.	.	.	.	.

7) Construct C programs to print the following patterns using loop statement.

```
1
22
333
4444
55555
```

```
*
# #
* * *
# # # #
* * * * *
```

```
1
0 1
1 0 1
0 1 0 1
```

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

```
    1
  A   B
2   3   4
C   D   E   F
```

```
* * * * *
*       *
*       *
* * * * *
```

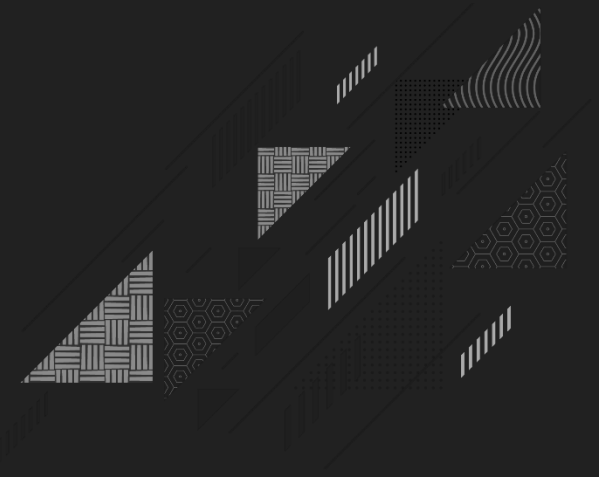
```
* * * * *
* * * *
* * *
* *
*
```

Thank you

Array & Strings

USING

{C}
Programming



Need of Array Variable

- Suppose we need to store `rollno` of the student in the integer variable.

Declaration

```
int rollno;
```

- Now we need to store `rollno` of 100 students.

Declaration

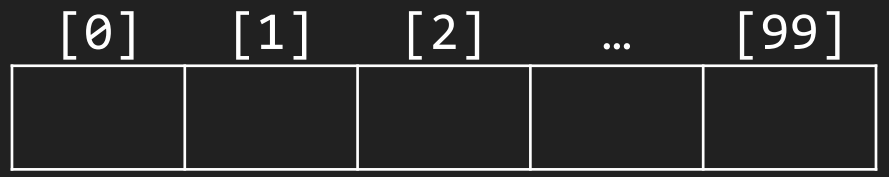
```
int rollno101, rollno102, rollno103, rollno104...;
```

- This is not appropriate to declare these many integer variables.
  e.g. 100 integer variables for `rollno`.
- Solution to declare and store multiple variables of similar type is an array.
- An array is a variable that can store multiple values.

Definition: Array

- An array is a fixed size sequential collection of elements of same data type grouped under single variable name.

```
int rollno[100];
```



Fixed Size	Sequential	Same Data type	Single Name
Here, the size of an array is 100 (fixed) to store rollno	It is indexed to 0 to 99 in sequence	All the elements (0-99) will be integer variables	All the elements (0-99) will be referred as a common name rollno

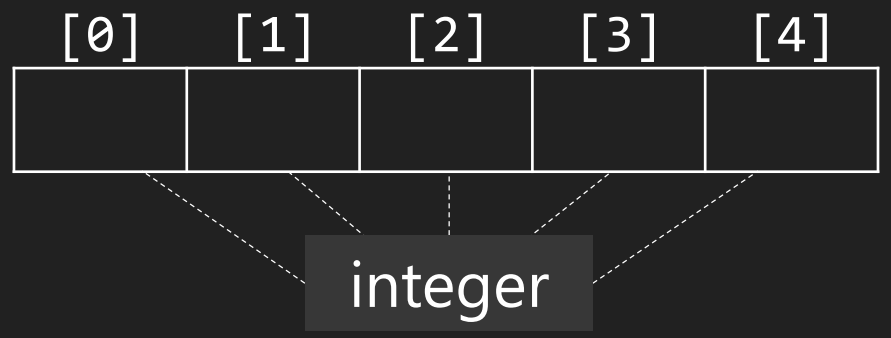
Declaring an array

Syntax

```
data-type variable-name[size];
```

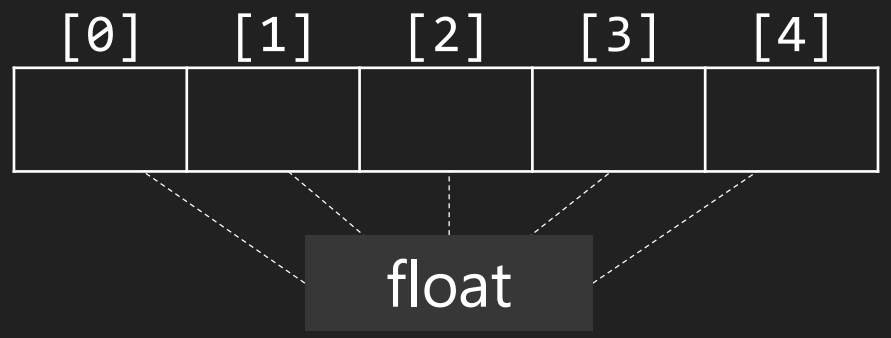
Integer Array

```
int mark[5];
```



Float Array

```
float avg[5];
```



- By default array index starts with `0`.
- If we declare an array of size `5` then its index ranges from `0` to `4`.
- First element will be store at `mark[0]` and last element will be stored at `mark[4]` not `mark[5]`.
- Like integer and float array we can declare array of type `char`.

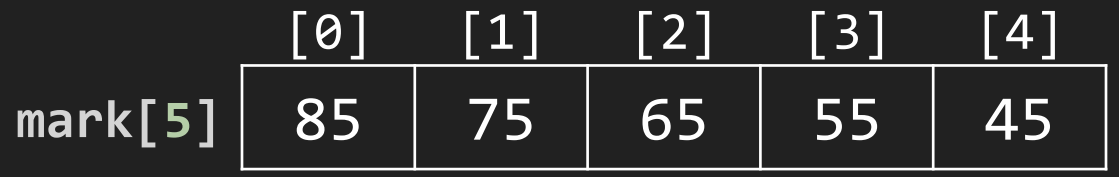
Initialing and Accessing an Array

Declaring, initializing and accessing single integer variable

```
int mark=90;      //variable mark is initialized with value 90
printf("%d",mark); //mark value printed
```

Declaring, initializing and accessing integer array variable

```
int mark[5]={85,75,76,55,45}; //mark is initialized with 5 values
printf("%d",mark[0]); //prints 85
printf("%d",mark[1]); //prints 75
printf("%d",mark[2]); //prints 65
printf("%d",mark[3]); //prints 55
printf("%d",mark[4]); //prints 45
```



Read(Scan) Array Elements

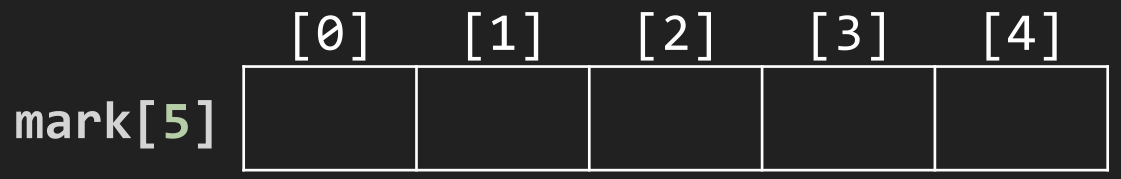
Reading array without loop

```c
void main()
{
  int mark[5];
  printf("Enter array element=");
  scanf("%d",&mark[0]);
  printf("Enter array element=");
  scanf("%d",&mark[1]);
  printf("Enter array element=");
  scanf("%d",&mark[2]);
  printf("Enter array element=");
  scanf("%d",&mark[3]);
  printf("Enter array element=");
  scanf("%d",&mark[4]);

  printf("%d",mark[0]);
  printf("%d",mark[1]);
  printf("%d",mark[2]);
  printf("%d",mark[3]);
  printf("%d",mark[4]);
}
```

Reading array using loop

```c
void main()
{
  int mark[5],i;
  for(i=0;i<5;i++)
  {
   printf("Enter array element=");
   scanf("%d",&mark[i]);
  }
  for(i=0;i<5;i++)
  {
   printf("%d",mark[i]);
  }
}
```



Develop a program to count number of positive or negative number from an array of 10 numbers.

Program

```c
void main(){
    int num[10],i,pos,neg;
    pos = 0;
    neg = 0;
    for(i=0;i<10;i++)
    {
        printf("Enter array element=");
        scanf("%d",&num[i]);
    }
    for(i=0;i<10;i++)
    {
        if(num[i]>0)
            pos=pos+1;
        else
            neg=neg+1;
    }
    printf("Positive=%d,Negative=%d",pos,neg);
}
```

Output

```
Enter array element=1
Enter array element=2
Enter array element=3
Enter array element=4
Enter array element=5
Enter array element=-1
Enter array element=-2
Enter array element=3
Enter array element=4
Enter array element=5
Positive=8,Negative=2
```

Develop a program to read n numbers in an array and print them in reverse order.

Program

```c
void main()
{
    int num[100],n,i;
    printf("Enter number of array elements=");
    scanf("%d",&n);
//loop will scan n elements only
    for(i=0;i<n;i++)
    {
        printf("Enter array element=");
        scanf("%d",&num[i]);
    }
//negative loop to print array in reverse order
    for(i=n-1;i>=0;i--)
    {
        printf("%d\n",num[i]);
    }
}
```

Output

```
Enter number of array
elements=5
Enter array element=1
Enter array element=2
Enter array element=3
Enter array element=4
Enter array element=5
5
4
3
2
1
```

Practice Programs

1) Develop a program to calculate sum of n array elements in C.
2) Develop a program to calculate average of n array elements in C.
3) Develop a program to find largest array element in C.
4) Develop a program to print sum of second and second last element of an array.
5) Develop a program to copy array elements to another array.
6) Develop a program to count odd and even elements of an array.

Multi Dimensional Array

Declaring 2 Dimensional Array

Syntax

```
data-type variable-name[x][y];
```

Declaration

```
int data[3][3]; //This array can hold 9 elements
```

```
int data[3][3];
```

	Column-0	Column-1	Column-2
Row-0	data[0][0]	data[0][1]	data[0][2]
Row-1	data[1][0]	data[1][1]	data[1][2]
Row-2	data[2][0]	data[2][1]	data[2][2]

- A two dimensional array can be seen as a table with ‘x’ rows and ‘y’ columns.
- The row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1).

Initialing and Accessing a 2D Array: Example-1

Program

```
1  int data[3][3] = {
2  {1,2,3}, //row 0 with 3 elements
3  {4,5,6}, //row 1 with 3 elements
4  {7,8,9}  //row 2 with 3 elements
5      };
6  printf("%d",data[0][0]); //1
7  printf("%d",data[0][1]); //2
8  printf("%d\n",data[0][2]); //3
9
10 printf("%d",data[1][0]); //4
11 printf("%d",data[1][1]);  //5
12 printf("%d\n",data[1][2]);  //6
13
14 printf("%d",data[2][0]);//7
15 printf("%d",data[2][1]); //8
16 printf("%d",data[2][2]); //9
```

```
1  // data[3][3] can be initialized like this also
2  int data[3][3]={{1,2,3},{4,5,6},{7,8,9}};
```

	Column-0	Column-1	Column-2
Row-0	1	2	3
Row-1	4	5	6
Row-2	7	8	9

Initialing and Accessing a 2D Array: Example-2

Program

```
int data[2][4] = {
{1,2,3,4}, //row 0 with 4 elements
{5,6,7,8}, //row 1 with 4 elements
    };
printf("%d",data[0][0]); //1
printf("%d",data[0][1]); //2
printf("%d",data[0][2]); //3
printf("%d\n",data[0][3]); //4

printf("%d",data[1][0]); //5
printf("%d",data[1][1]); //6
printf("%d",data[1][2]); //7
printf("%d",data[1][3]); //8
```

```
// data[2][4] can be initialized like this also
int data[2][4]={{1,2,3,4},{5,6,7,8}};
```

	Col-0	Col-1	Col-2	Col-3
Row-0	1	2	3	4
Row-1	5	6	7	8

Read(Scan) 2D Array Elements

Program

```
1  void main(){
2      int data[3][3],i,j;
3      for(i=0;i<3;i++)
4      {
5          for(j=0;j<3;j++)
6          {
7              printf("Enter array element=");
8              scanf("%d",&data[i][j]);
9          }
10     }
11     for(i=0;i<3;i++)
12     {
13         for(j=0;j<3;j++)
14         {
15             printf("%d",data[i][j]);
16         }
17         printf("\n");
18     }
19 }
```

	Column-0	Column-1	Column-2
Row-0	1	2	3
Row-1	4	5	6
Row-2	7	8	9

Output

```
Enter array element=1
Enter array element=2
Enter array element=3
Enter array element=4
Enter array element=5
Enter array element=6
Enter array element=7
Enter array element=8
Enter array element=9
123
456
789
```

Develop a program to count number of positive, negative and zero elements from 3 X 3 matrix

Program

```c
void main(){
    int data[3][3],i,j,pos=0,neg=0,zero=0;
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("Enter array element=");
            scanf("%d",&data[i][j]);
            if(data[i][j]>0)
                pos=pos+1;
            else if(data[i][j]<0)
                neg=neg+1;
            else
                zero=zero+1;
        }
    }
    printf("positive=%d,negative=%d,zero=%d",pos,neg,zero);
}
```

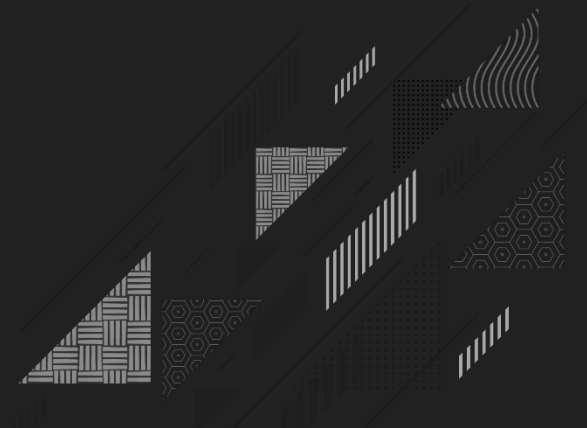
Output

```
Enter array element=9
Enter array element=5
Enter array element=6
Enter array element=-3
Enter array element=-7
Enter array element=0
Enter array element=11
Enter array element=13
Enter array element=8
positive=6,negative=2,zero=1
```

Practice Programs

1. Develop a program to perform addition of two matrix.
2. Develop a program to perform multiplication of two matrix.

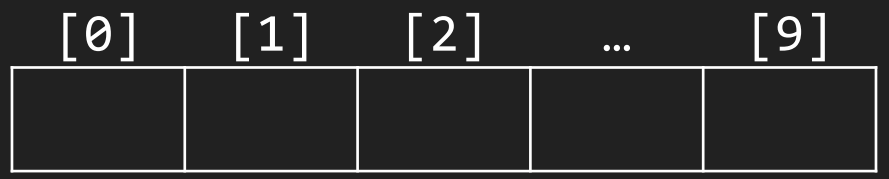
String (Character Array)



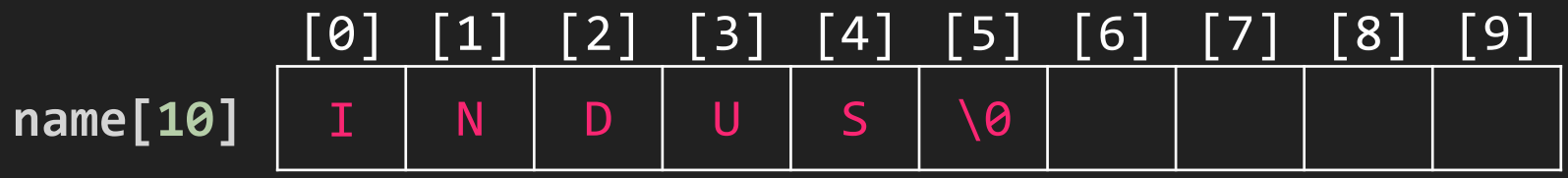
Definition: String

- A String is a one-dimensional array of characters terminated by a `null('\0')`.

```
char name[10];
```



- Each character in the array occupies one byte of memory, and the last character must always be `null('\0')`.
- The termination character `('\0')` is important in a string to identify where the string ends.



Declaring & Initializing String

Declaration

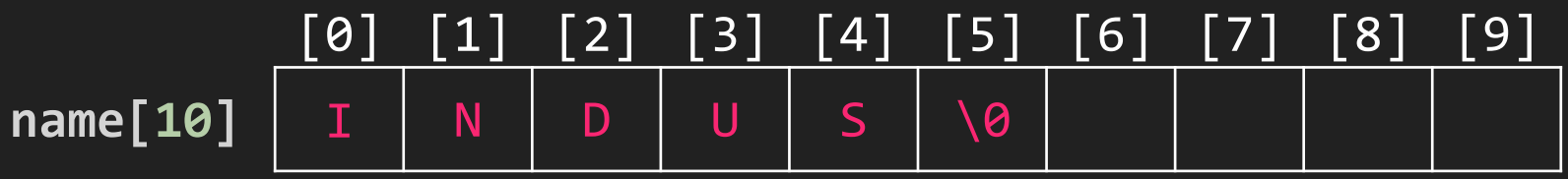
```
char name[10];
```

Initialization method 1:

```
char name[10]={‘I',’N',’D',’U',’S','\0'};
```

Initialization method 2:

```
char name[10]=“INDUS";
//'\0' will be automatically inserted at the end in this type of declaration.
```



Read String: scanf()

Program

```c
void main()
{
    char name[10];
    printf("Enter name:");
    scanf("%s",name);
    printf("Name=%s",name);
}
```

Output

```
Enter name: Indus
Name=Indus
```

Output

```
Enter name: CE IT CSE
Name=CE
```

- There is no need to use address of `(&)` operator in `scanf` to store a string.
- As string `name` is an array of characters and the name of the array, i.e., `name` indicates the base address of the string (character array).
- `scanf()` terminates its input on the first whitespace(space, tab, newline etc.) encountered.

Read String: gets()

Program

```c
#include<stdio.h>
void main()
{
    char name[10];
    printf("Enter name:");
    gets(name); //read string including white spaces
    printf("Name=%s",name);
}
```

Output

```
Enter name:Indus Institute
Name=Indus Institute
```

- `gets()`: Reads characters from the standard input and stores them as a string.
- `puts()`: Prints characters from the standard.
- `scanf()`: Reads input until it encounters whitespace, newline or End Of File(`EOF`) whereas `gets()` reads input until it encounters newline or End Of File(`EOF`).
- `gets()`: Does not stop reading input when it encounters whitespace instead it takes whitespace as a string.

String Handling Functions : strlen()

- C has several inbuilt functions to operate on string. These functions are known as string handling functions.
- `strlen(s1):` returns length of a string in integer

Program

```c
#include <stdio.h>
#include <string.h> //header file for string functions
void main()
{
    char s1[10];
    printf("Enter string:");
    gets(s1);
    printf("%d",strlen(s1)); // returns length of s1 in integer
}
```

Output

```
Enter string: CE CSE IT
09
```

String Handling Functions: strcmp()

- `strcmp(s1,s2):` Returns `0` if `s1` and `s2` are the same.
- Returns less than `0` if `s1<s2.`
- Returns greater than `0` if `s1>s2`.

Program

```
1  void main()
2  {
3      char s1[10],s2[10];
4      printf("Enter string-1:");
5      gets(s1);
6      printf("Enter string-2:");
7      gets(s2);
8      if(strcmp(s1,s2)==0)
9         printf("Strings are same");
10     else
11        printf("Strings are not same");
12 }
```

Output

```
Enter string-1:Computer
Enter string-2:Computer
Strings are same
```

Output

```
Enter string-1:Computer
Enter string-2:Computer
Strings are same
```

String Handling Functions

For examples consider: `char s1[]="Their",s2[]="There";`

Syntax	Description
`strcpy(s1,s2)`	Copies 2nd string to 1st string. `strcpy(s1,s2)` copies the string `s2` in to string `s1` so `s1` is now "There". `s2` remains unchanged.
`strcat(s1,s2)`	Appends 2nd string at the end of 1st string. `strcat(s1,s2);` a copy of string `s2` is appended at the end of string `s1`. Now `s1` becomes "TheirThere"
`strchr(s1,c)`	Returns a pointer to the first occurrence of a given character in the string `s1`. `printf("%s",strchr(s1,'i'));` Output : ir
`strstr(s1,s2)`	Returns a pointer to the first occurrence of a given string `s2` in string `s1`. `printf("%s",strstr(s1,"he"));` Output : heir

String Handling Functions (Cont...)

For examples consider: `char s1[]="Their",s2[]="There";`

Syntax	Description
`strrev(s1)`	Reverses given string. `strrev(s1);` makes string `s1` to "riehT"
`strlwr(s1)`	Converts string `s1` to lower case. `printf("%s",strlwr(s1));` Output : their
`strupr(s1)`	Converts string `s1` to upper case. `printf("%s",strupr(s1));` Output : THEIR

`strncpy(s1,s2,n)`	Copies first n character of string `s2` to string `s1`
	`s1=""; s2="There";`
	`strncpy(s1,s2,2);`
	`printf("%s",s1);`
	Output : Th

`strncat(s1,s2,n)`	Appends first n character of string `s2` at the end of string `s1`.
	`strncat(s1,s2,2);`
	`printf("%s", s1);`
	Output : TheirTh

String Handling Functions (Cont...)

For examples consider: `char s1[]="Their",s2[]="There";`

Syntax	Description
`strncmp(s1,s2,n)`	Compares first n character of string `s1` and `s2` and returns similar result as `strcmp()` function. `printf("%d",strcmp(s1,s2,3));` Output : 0
`strrchr(s1,c)`	Returns the last occurrence of a given character in a string `s1`. `printf("%s",strrchr(s2,'e'));` Output : ere

Thank you

Programming for Problem Solving (PPS)

USING

{C}
Programming

Functions

What is Function?

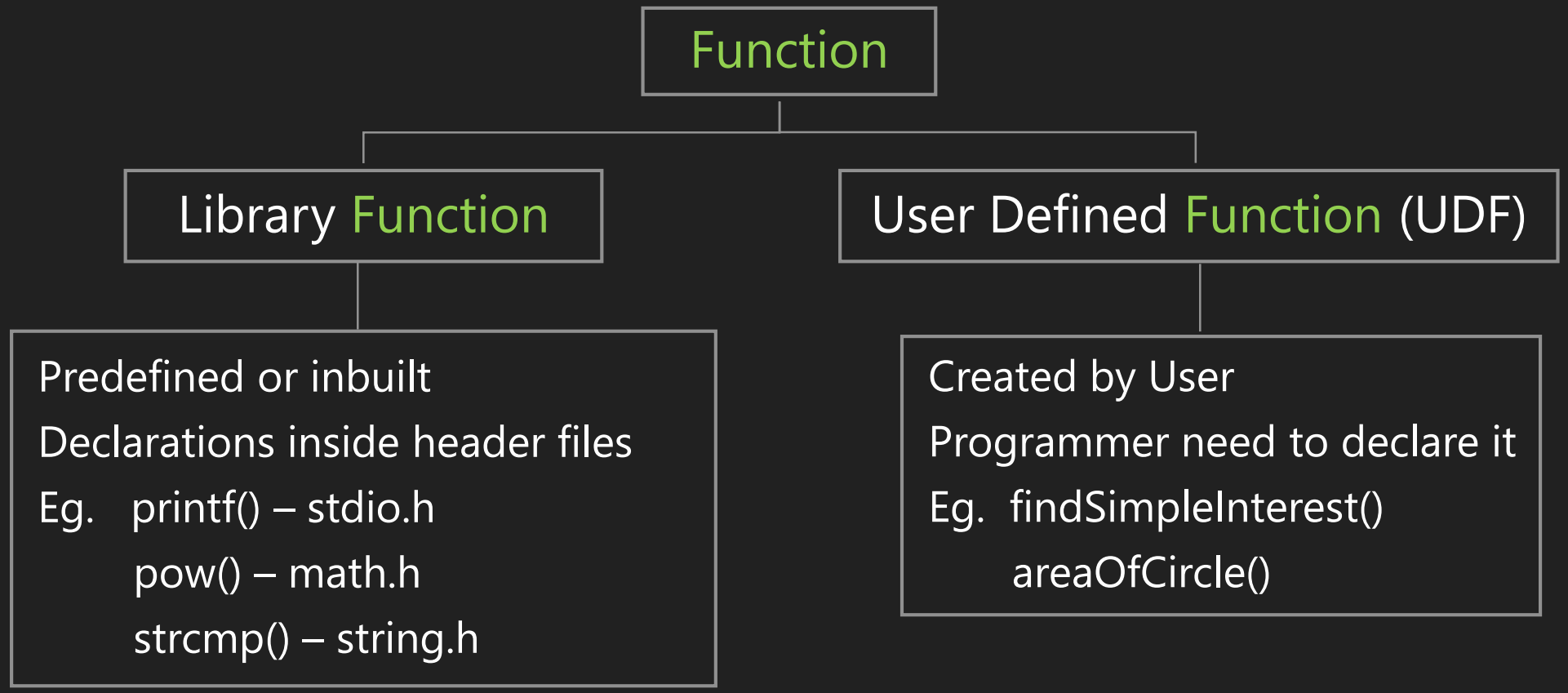
- A function is a group of statements that perform a specific task.
- It divides a large program into smaller parts.
- A function is something like hiring a person to do a specific job for you.
- Every C program can be thought of as a collection of these functions.
- Program execution in C language starts from the main function.

Syntax

```
void main()
{
    // body part
}
```

- Why function ?
  - Avoids rewriting the same code over and over.
  - Using functions it becomes easier to write programs and keep track of what they doing.

Types of Function



Program Structure for Function

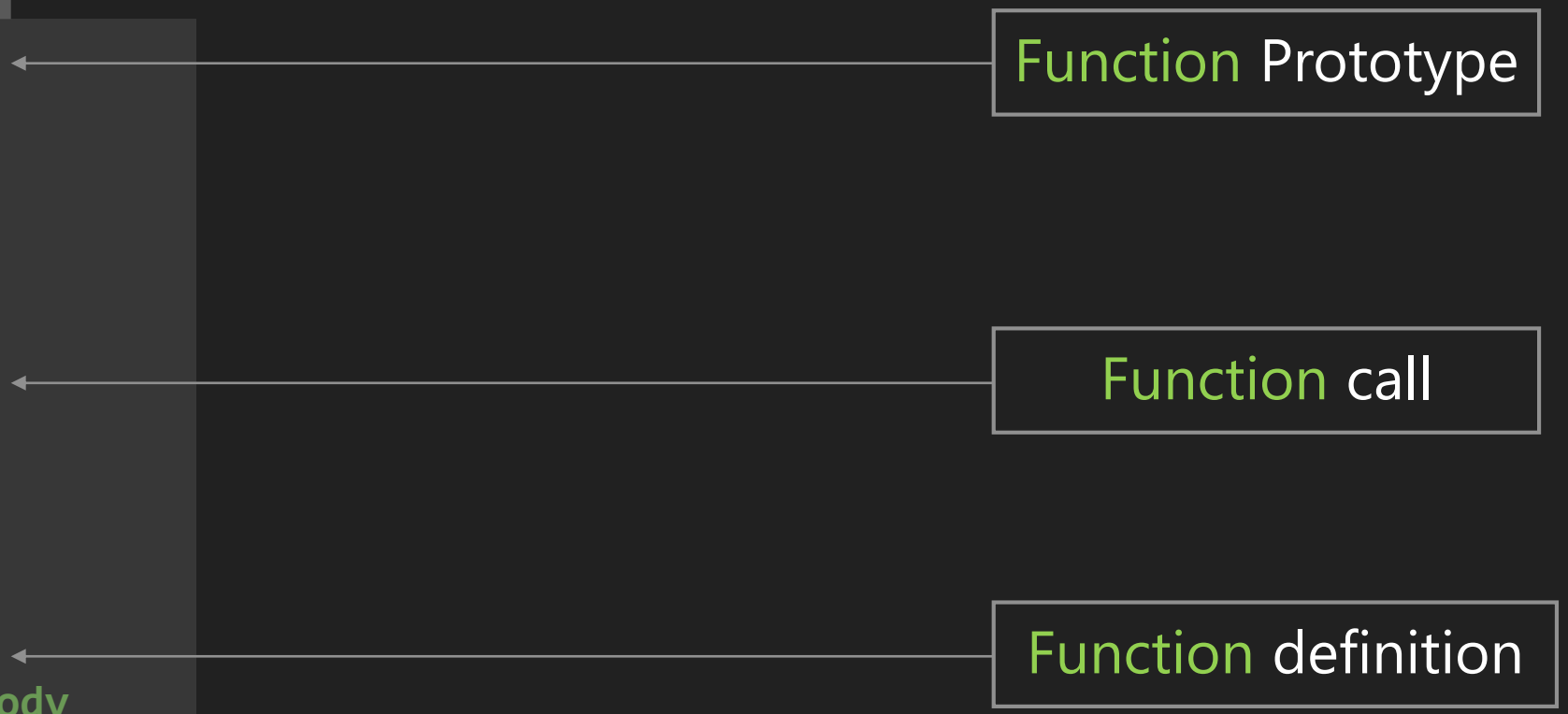
- When we use a user-defined function program structure is divided into three parts.

Function Structure

```
void func1();


void main()
{
  ....
  func1();
}

void func1()
{
  ....
  //function body
  ....
}
```



Function Prototype

- A function Prototype also know as function declaration.
- A function declaration tells the compiler about a function name and how to call the function.
- It defines the function before it is being used or called.
- A function prototype needs to be written at the beginning of the program.

Syntax

```
return-type function-name (arg-1, arg 2, …);
```

Example

```
void addition(int, int);
```

Function Definition

- A function definition defines the functions header and body.
- A function header part should be identical to the function prototype.
  - Function return type
  - Function name
  - List of parameters
- A function body part defines function logic.
  - Function statements

Syntax

```
return-type function-name (arg-1, arg 2, …)
{
        //... Function body
}
```

Example

```
void addition(int x, int y)
{
 printf("Addition is=%d“,(x+y));
}
```

WAP to add two number using add(int, int) Function

Program

```c
#include <stdio.h>
void add(int, int); // function declaration

void main()
{
    int a = 5, b = 6;
    add(a, b); // function call
}

void add(int x, int y) // function definition
{
   printf("Addition is = %d", x + y);
}
```

Output

```
Addition is = 11
```

Actual parameters and Formal parameters

- Values that are passed to the called function from the main function are known as Actual parameters.
- The variables declared in the function prototype or definition are known as Formal parameters.
- When a method is called, the formal parameter is temporarily "bound" to the actual parameter.

Actual parameters

```
void main()
{
    int a = 5, b = 6;
    add(a, b); // a and b are the
    actual parameters in this call.
}
```

Formal parameters

```
void add(int x, int y) // x and y are
formal parameters.
{
   printf("Addition is = %d", x + y);

}
```

Return Statement

- If function is returning a value to calling function, it needs to use the keyword **return**.
- The called function can only return one value per call.

Syntax

```
return;
            Or

return (expression);
```

WAP to find maximum number from two number

Program

```c
#include <stdio.h>
int max(int a, int b);
void main()
{
    int a = 100;
    int b = 200;
    int maxvalue;
    maxvalue = max(a, b);
    printf("Max value is : %d\n",
    maxvalue);
}
int max(int a, int b)
{
    if (a > b)
        return a; // return a
    else
        return b; // return b
}
```

Output

```
Max value is : 200
```

WAP to calculate the Power of a Number

Program

```c
#include <stdio.h>
int power(int, int);
void main()
{
    int num, pow, res;
    printf("Enter any number : ");
    scanf("%d", &num);
    printf("Enter power of number : ");
    scanf("%d", &pow);
    res = power(num, pow);
    printf("%d's power %d = %d", num, pow, res);
}
int power(int n, int p)
{   int r = 1;
    while (p >= 1)
    {
        r = r * n;
        p--;
    }
    return r;}
```

Output

```
Enter any number : 5
Enter power of number : 3
5's power 3 = 125
```

WAP to find Factorial of a Number

Program

```c
#include <stdio.h>
int fact(int);
int main()
{
    int n, f;
    printf("Enter the number :\n");
    scanf("%d", &n);
    f = fact(n);
    printf("factorial = %d", f);
}
int fact(int n)
{
    int i, fact = 1;
    for (i = 1; i <= n; i++)
        fact = fact * i;
    return fact;
}
```

Output

```
Enter the number :
5
factorial = 120
```

WAP to check Number is Prime or not

Program

```c
#include <stdio.h>
int checkPrime(int);
void main()
{
    int n1, prime;
    printf("Enter the number :");
    scanf("%d", &n1);
    prime = checkPrime(n1);
    if (prime == 1)
        printf("The number %d is a prime
        number.\n", n1);
    else
        printf("The number %d is not a prime
        number.\n", n1);
}
```

Program contd.

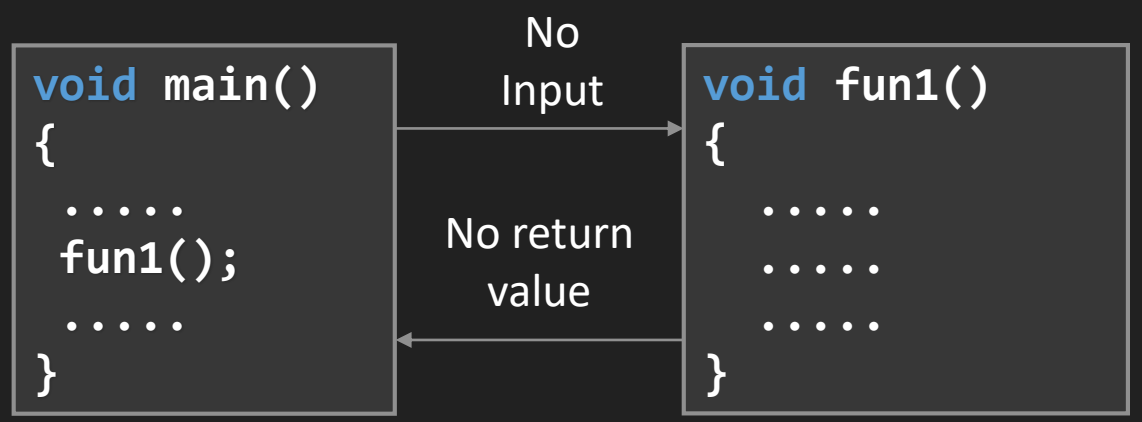
```c
int checkPrime(int n1)
{
    int i = 2;
    while (i <= n1 / 2)
    {
        if (n1 % i == 0)
            return 0;
        else
            i++;
    }
    return 1;
}
```

Output

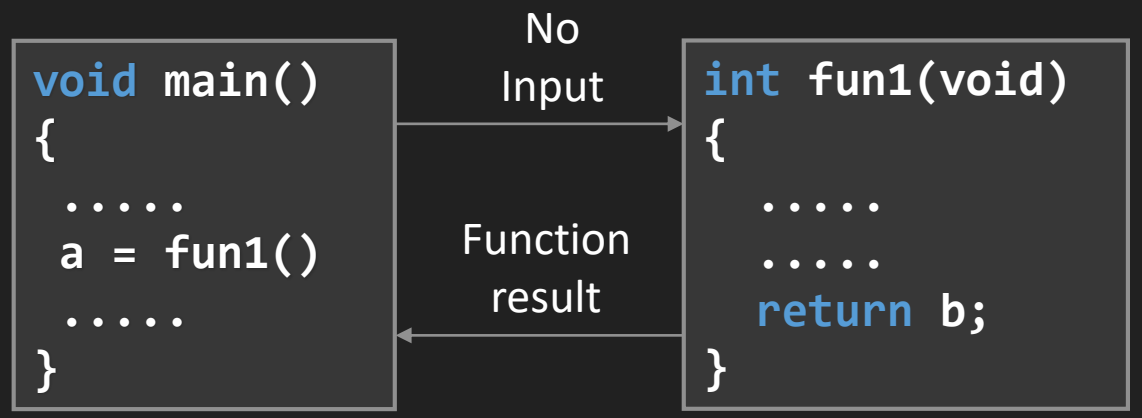
```
Enter the number :7
The number 7 is a prime number.
```

Category of Function

(1) Function with no argument and but no return value

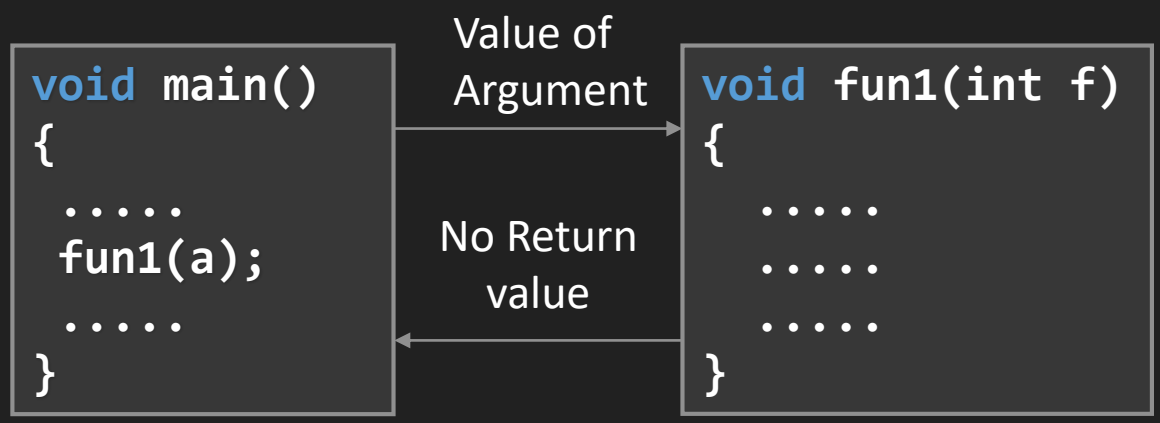


(2) Function with no argument and returns value

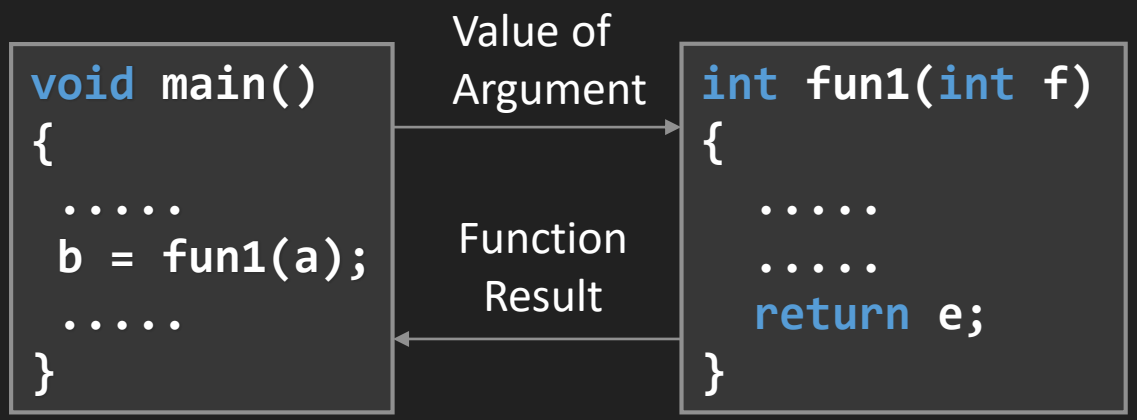


Category of Function cont.

(3) Function with argument and but no return value



(4) Function with argument and returns value



Advantages of Function

- Using function we can avoid rewriting the same logic or code again and again in a program.
- We can track or understand large program easily when it is divide into functions.
- It provides reusability.
- It help in testing and debugging because it can be tested for errors individually in the easiest way.
- Reduction in size of program due to code of a function can be used again and again, by calling it.

Practice Programs

1) WAP to count simple interest using function.
2) WAP that defines a function to add first *n* numbers.
3) WAP using global variable, static variable.
4) WAP that will scan a character string passed as an argument and convert all lowercase character into their uppercase equivalents.
5) Build a function to check number is prime or not. If number is prime then function return value 1 otherwise return 0.
6) Write a program to calculate nCr using user defined function. nCr = n! / (r! * (n-r)!)
7) Create a function to swap the values of two variables.
8) Write a function which takes 2 numbers as parameters and returns the gcd of the 2 numbers. Call the function in main().

Thank you

Programming for Problem Solving (PPS)

Recursion

USING

{C}
Programming

What is Recursion?

- Any function which calls itself is called recursive function and such function calls are called recursive calls.
- Recursion cannot be applied to all problems, but it is more useful for the tasks that can be defined in terms of a similar subtask.
- It is idea of representing problem a with smaller problems.
- Any problem that can be solved recursively can be solved iteratively.
- When recursive function call itself, the memory for called function allocated and different copy of the local variable is created for each function call.
- Some of the problem best suitable for recursion are
  - Factorial
  - Fibonacci
  - Tower of Hanoi

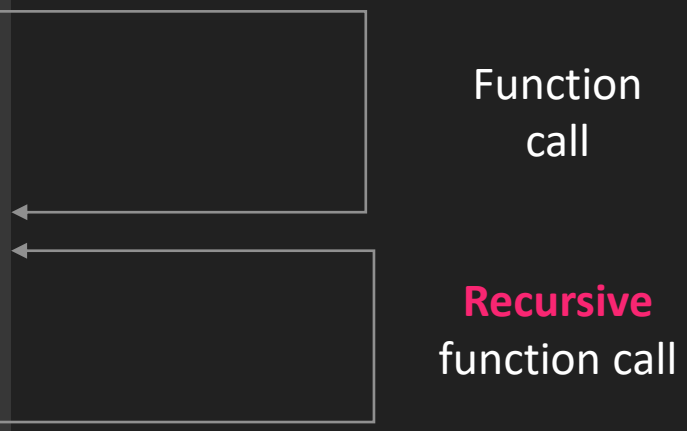
Working of Recursive function

Working

```
void func1();


void main()
{
  ....
  func1();
  ....
}

void func1()
{
  ....
  func1();
  ....
}
```



Function call

Recursive function call

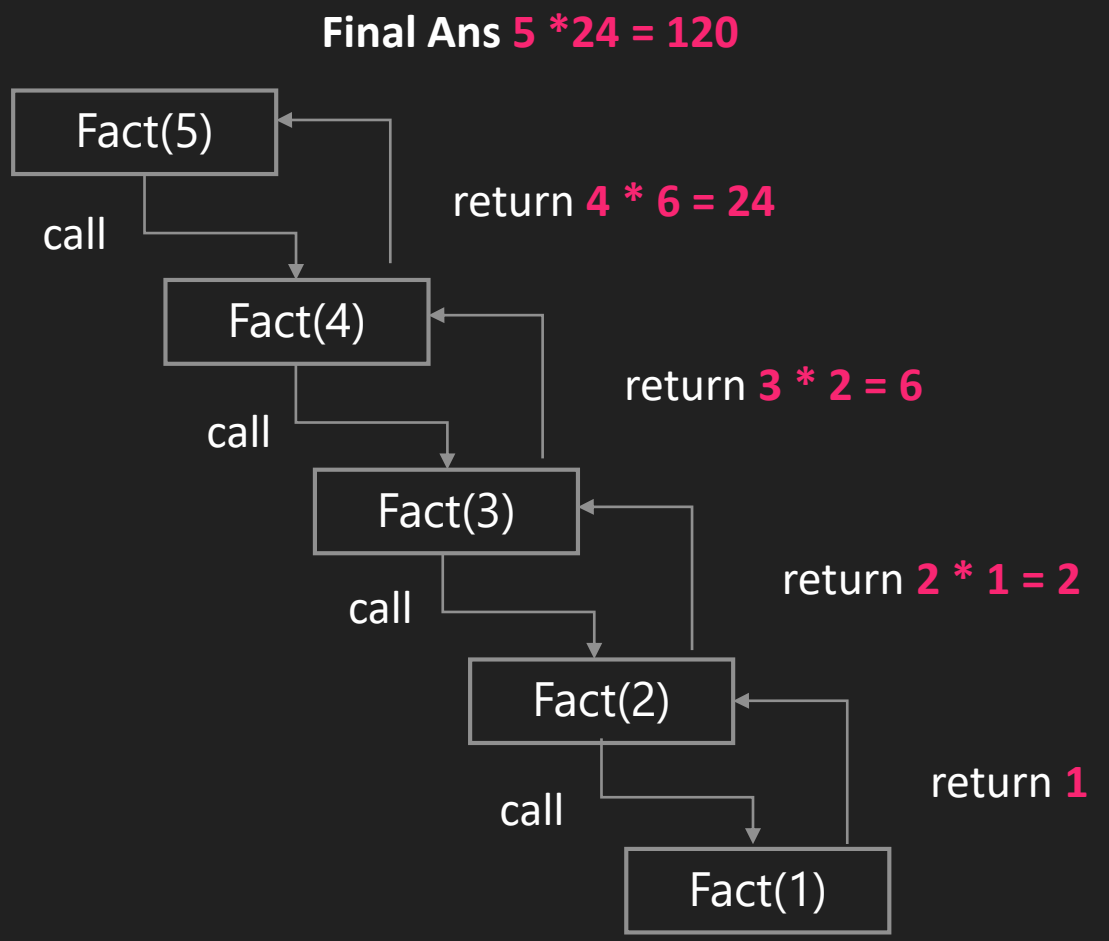
Properties of Recursion

- A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have.
- Base Case or Base criteria
  - It allows the recursion algorithm to stop.
  - A base case is typically a problem that is small enough to solve directly.
- Progressive approach
  - A recursive algorithm must change its state in such a way that it moves forward to the base case.

Recursion - factorial example

- The factorial of a integer n, is product of
  - n * (n-1) * (n-2) * …. * 1
- Recursive definition of factorial
  - n! = n * (n-1)!
  - Example
    - 3! = 3 * 2 * 1
    - 3! = 3 * (2 * 1)
    - 3! = 3 * (2!)

Recursive trace



WAP to find factorial of given number using Recursion

Program

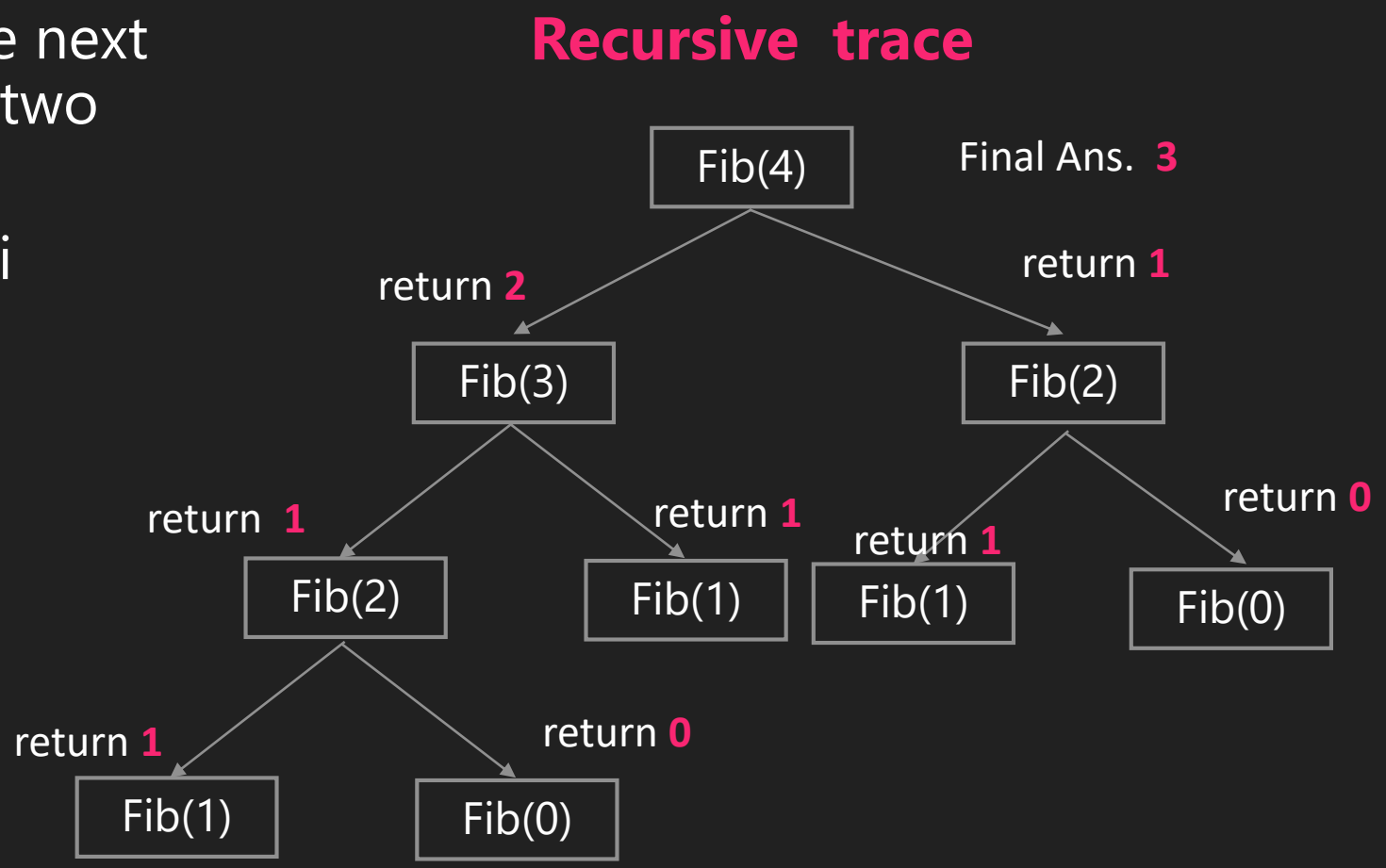
```c
#include <stdio.h>
int fact(int);
void main()
{
    int n, f;
    printf("Enter the number?\n");
    scanf("%d", &n);
    f = fact(n);
    printf("factorial = %d", f);
}
int fact(int n)
{
    if (n == 0)
        return 1;
    else if (n == 1)
        return 1;
    else
        return n * fact(n - 1);
}
```

Output

```
Enter the number? 5
factorial = 120
```

Recursion - Fibonacci example

- A series of numbers , where next number is found by adding the two number before it.
- Recursive definition of Fibonacci
  - Fib(0) = 0
  - Fib(1) = 1
  - Fib(n) = Fib(n-1) + Fib(n-2)
- Example
  - Fib(4) = Fib(3) + Fib(2)
  - Fib(4) = 3



WAP to Display Fibonacci Sequence

Program

```c
#include <stdio.h>
int fibonacci(int);
void main()
{
    int n, m = 0, i;
    printf("Enter Total terms\n");
    scanf("%d", &n);
    printf("Fibonacci series\n");
    for (i = 1; i <= n; i++)
    {
        printf("%d ", fibonacci(m));
        m++;
    }
}
```

Program contd.

```c
int fibonacci(int n)
{
    if (n == 0 || n == 1)
        return n;
    else
        return (fibonacci(n - 1) +
    fibonacci(n - 2));
}
```

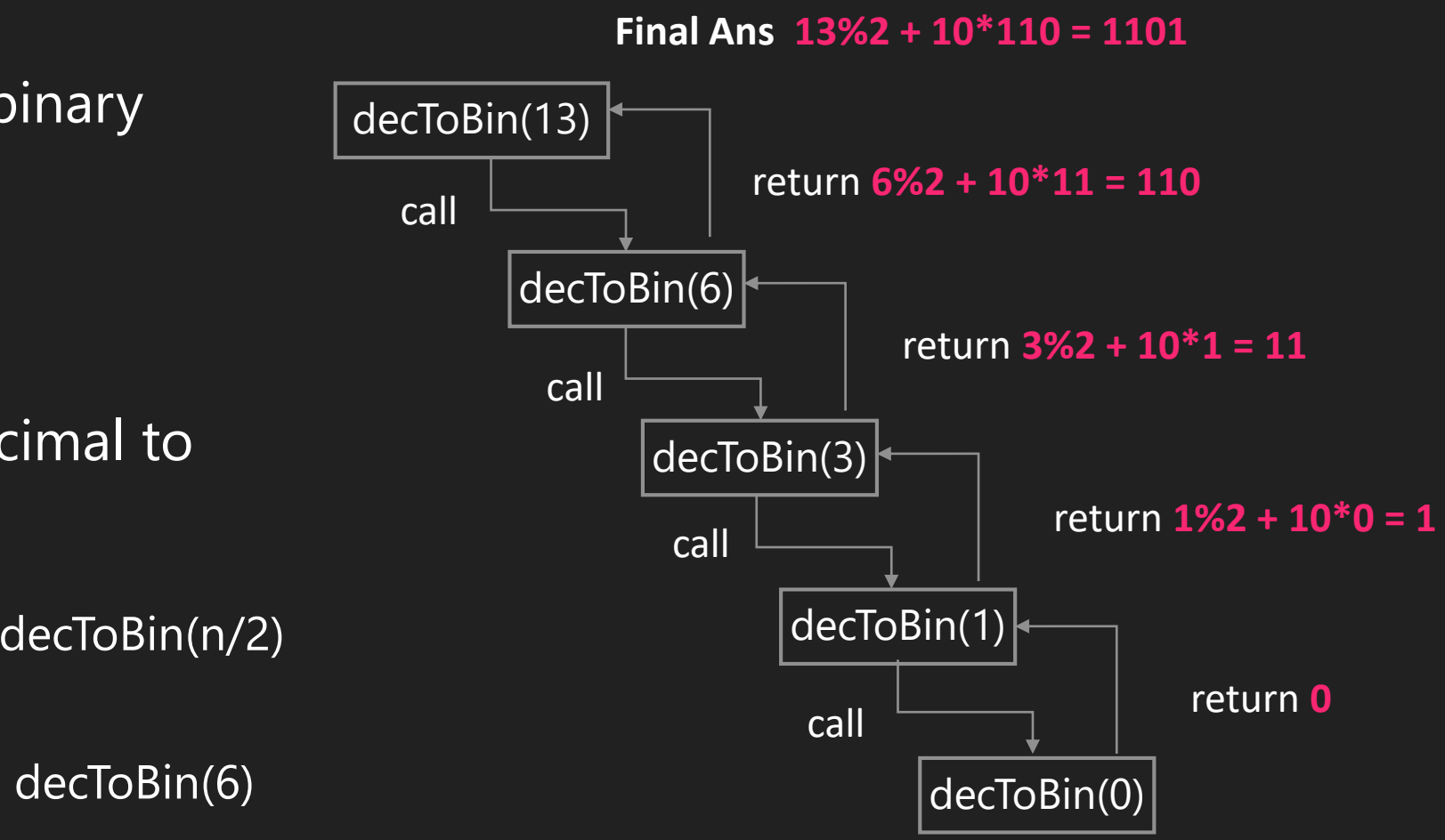
Output

```
Enter Total terms
5
Fibonacci series
0 1 1 2 3
```

Recursion - Decimal to Binary example

- To convert decimal to binary, divide decimal number by 2 till dividend will be less then 2
- To convert decimal 13 to binary
  - 13/2 = 6 reminder 1
  - 6/2 = 6 reminder 0
  - 3/2 = 3 reminder 1
  - 1/2 = 1 reminder 1
- Recursive definition of Decimal to Binary
  - decToBin(0) = 0
  - decToBin(n) = n%2 + 10* decToBin(n/2)
- Example
  - decToBin(13) = 13%2 + 10 decToBin(6)
  - decToBin(13) = 1101

Recursive trace



WAP to Convert Decimal to Binary

Program

```c
#include <stdio.h>
int convertDecimalToBinary(int);
void main()
{
    int dec, bin;
    printf("Enter a decimal number: ");
    scanf("%d", &dec);
    bin = convertDecimalToBinary(dec);
    printf("The binary equivalent = %d \n",bin);
}
int convertDecimalToBinary(int dec)
{
    if (dec == 0)
        return 0;
    else
        return (dec % 2 + 10 *
    convertDecimalToBinary(dec / 2));
}
```

Output

```
Enter a decimal number: 12
The binary equivalent = 1100
```

WAP to Convert Binary to Decimal

Program

```c
#include <stdio.h>
int convertBinaryToDecimal(int b, int c, int t);
void main()
{
    unsigned int binary, decimal;
    printf("Enter a binary number: ");
    scanf("%d", &binary);
    decimal = convertBinaryToDecimal(binary, 1, 0);
    printf("Decimal value of %d is %d", binary, decimal);
}
int convertBinaryToDecimal(int b, int c, int t)
{
    if (b > 0)
    {
        t += (b % 10) * c;
        convertBinaryToDecimal(b / 10, c * 2, t);
    }
    else
        return t;
}
```

Output

```
Enter a binary number: 101
Decimal value of 101 is 5
```

Practice Programs

1) Write a program to find factorial of a given number using recursion.
2) WAP to convert decimal number into binary using recursion.
3) WAP to use recursive calls to evaluate $F(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots + x^n/n!$

Thank you

Pointer

USING

{C}

Programming

What is Pointer?

- A normal variable is used to store value.
- A pointer is a variable that store address / reference of another variable.
- Pointer is derived data type in C language.
- A pointer contains the memory address of that variable as their value. Pointers are also called address variables because they contain the addresses of other variables.

Declaration & Initialization of Pointer

Syntax

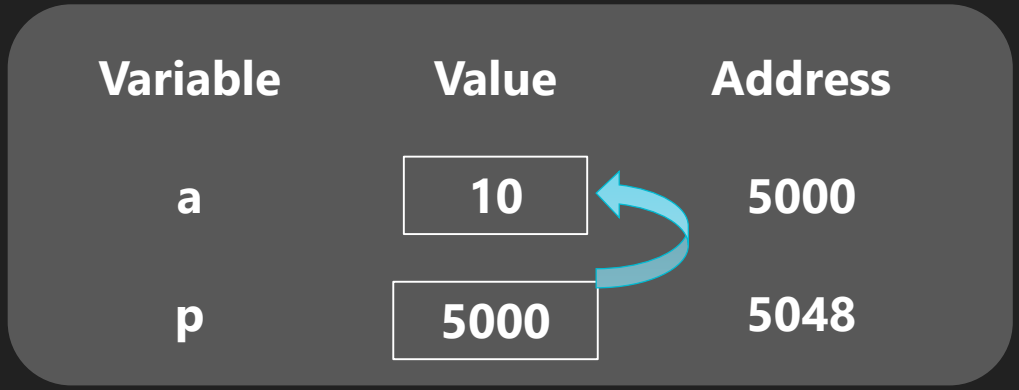
```
datatype *ptr_variablename;
```

Output

```
10 10 5000
```

Example

```
void main()
{
    int a=10, *p; // assign memory address of a
    to pointer variable p
    p = &a;
    printf("%d %d %d", a, *p, p);
}
```



- p is integer pointer variable
- & is address of or referencing operator which returns memory address of variable.
- * is indirection or dereferencing operator which returns value stored at that memory address.
- & operator is the inverse of * operator
- x = a is same as x = *(&a)

Why use Pointer?

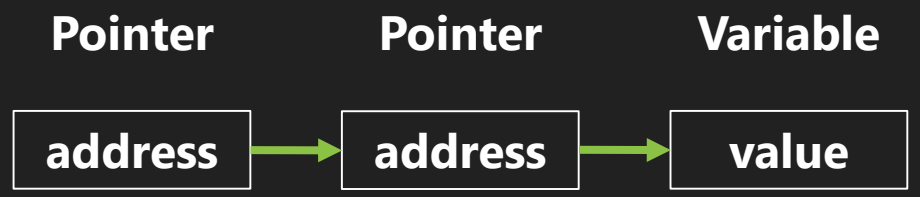
- C uses pointers to create dynamic data structures, data structures built up from blocks of memory allocated from the heap at run-time. Example linked list, tree, etc.
- C uses pointers to handle variable parameters passed to functions.
- Pointers in C provide an alternative way to access information stored in arrays.
- Pointer use in system level programming where memory addresses are useful. For example shared memory used by multiple threads.
- Pointers are used for file handling.
- This is the reason why C is versatile.

Pointer to Pointer – Double Pointer

- Pointer holds the address of another variable of same type.
- When a pointer holds the address of another pointer then such type of pointer is known as pointer-to-pointer or double pointer.
- The first pointer contains the address of the second pointer, which points to the location that contains the actual value.

Syntax

```
datatype **ptr_variablename;
```



Example

```
int **ptr;
```

Write a program to print variable, address of pointer variable and pointer to pointer variable.

Program

```c
#include <stdio.h>
int main () {
    int var;
    int *ptr;
    int **pptr;
    var = 3000;
    ptr = &var; // address of var
    pptr = &ptr; // address of ptr using address of operator &
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
        printf("Value available at **pptr = %d\n", **pptr);
        return 0;
}
```

Output

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Relation between Array & Pointer

- When we declare an array, compiler allocates continuous blocks of memory so that all the elements of an array can be stored in that memory.
- The address of first allocated byte or the address of first element is assigned to an array name.
- Thus array name works as pointer variable.
- The address of first element is also known as base address.

Relation between Array & Pointer – Cont.

- Example: `int a[10], *p;`
- a[0] is same as *(a+0), a[2] is same as *(a+2) and a[i] is same as *(a+i)

Array of Pointer

- As we have an array of char, int, float etc, same way we can have an array of pointer.
- Individual elements of an array will store the address values.
- So, an array is a collection of values of similar type. It can also be a collection of references of similar type known by single name.

Syntax

```
datatype *name[size];
```

Example

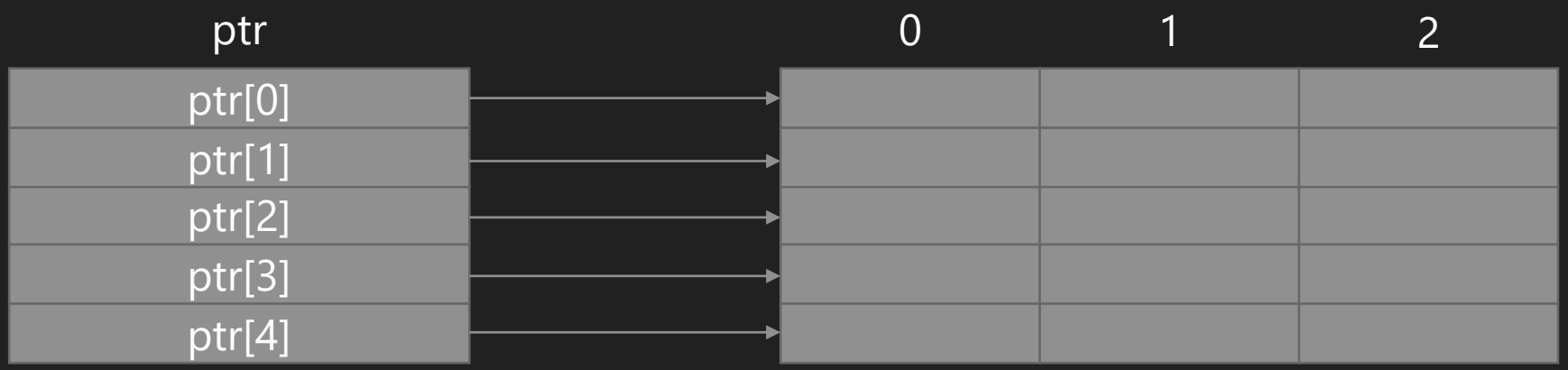
```
int *ptr[5]; //declares an array of integer pointer of size 5
```

Array of Pointer – Cont.

- An array of pointers ptr can be used to point to different rows of matrix as follow:

Example

```
1 for(i=0; i<5; i++)
2 {
3         ptr[i]=&mat[i][0];
4 }
```



- By dynamic memory allocation, we do not require to declare two-dimensional array, it can be created dynamically using array of pointers.

Write a program to swap value of two variables using pointer / call by reference.

Program

```c
int main()
{
    int num1,num2;
    printf("Enter value of num1 and num2: ");
    scanf("%d %d",&num1, &num2);

//displaying numbers before swapping
    printf("Before Swapping: num1 is: %d, num2 is: %d\n",num1,num2);

//calling the user defined function swap()
    swap(&num1,&num2);

//displaying numbers after swapping
    printf("After Swapping: num1 is: %d, num2 is: %d\n",num1,num2);
    return 0;
}
```

Output

```
Enter value of num1 and num2: 5
10
Before Swapping: num1 is: 5, num2 is: 10
After  Swapping: num1 is: 10, num2 is: 5
```

Pointer and Function

- Like normal variable, pointer variable can be passed as function argument and function can return pointer as well.
- There are two approaches to passing argument to a function:
  - Call by value
  - Call by reference / address

Call by Value

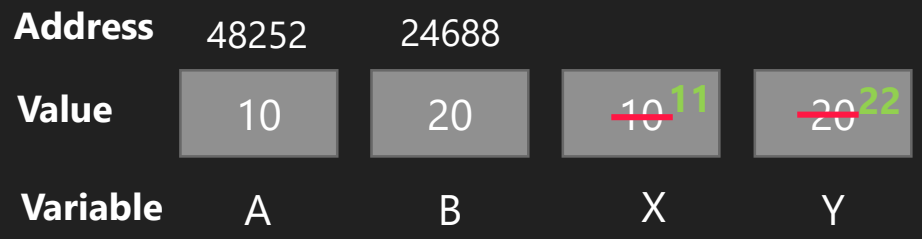
- In this approach, the values are passed as function argument to the definition of function.

Program

```
1  #include<stdio.h>
2  void fun(int,int);
3  int main()
4  {
5      int A=10,B=20;
6      printf("\nValues before calling %d, %d",A,B);
7      fun(A,B);
8      printf("\nValues after calling %d, %d",A,B);
9      return 0;
10 }
11 void fun(int X,int Y)
12 {
13     X=11;
14     Y=22;
15 }
```

Output

```
Values before calling 10, 20
Values after calling 10, 20
```



Call by Reference / Address

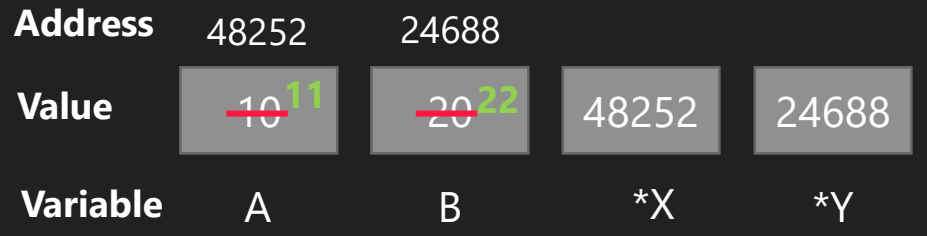
- In this approach, the references / addresses are passed as function argument to the definition of function.

Program

```
1  #include<stdio.h>
2  void fun(int*,int*);
3  int main()
4  {
5      int A=10,B=20;
6      printf("\nValues before calling %d, %d",A,B);
7      fun(&A,&B);
8      printf("\nValues after calling %d, %d",A,B);
9      return 0;
10 }
11 void fun(int *X,int *Y)
12 {
13     *X=11;
14     *Y=22;
15 }
```

Output

```
Values before calling 10, 20
Values after  calling 11, 22
```



Pointer to Function

- Every function has reference or address, and if we know the reference or address of function, we can access the function using its reference or address.
- This is the way of accessing function using pointer.

Syntax

```
return-type (*ptr-function)(argument list);
```

- return-type: Type of value function will return.
- argument list: Represents the type and number of value function will take, values are sent by the calling statement.
- (*ptr-function): The parentheses around *ptr-function tells the compiler that it is pointer to function.
- If we write *ptr-function without parentheses then it tells the compiler that ptr-function is a function that will return a pointer.

Write a program to sum of two numbers using pointer to function.

Program

```c
#include<stdio.h>
int Sum(int,int);
int (*ptr)(int,int);
int main()
{
    int a,b,rt;
    printf("\nEnter 1st number : ");
    scanf("%d",&a);
    printf("\nEnter 2nd number : ");
    scanf("%d",&b);
    ptr = Sum;
    rt = (*ptr)(a,b);
    printf("\nThe sum is : %d",rt);
    return 0;
}
int Sum(int x,int y)
{
        return x + y;
}
```

Output

```
Enter 1st number : 5

Enter 2nd number : 10

The sum is : 15
```

Practice Programs

1. Write a C program to print the address of variable using pointer.
2. Write a C a program to swap two elements using pointer.
3. Write a C a program to print value and address of a variable
4. Write a C a program to calculate sum of two numbers using pointer
5. Write a C a program to swap value of two numbers using pointer
6. Write a C a program to calculate sum of elements of an array using pointer
7. Write a C a program to swap value of two variables using function
8. Write a C a program to print the address of character and the character of string using pointer
9. Write a C a program for sorting using pointer

Thank you

Union

USING

{C}

Programming

What is Union?

- Union is a user defined data type similar like Structure.
- It holds different data types in the same memory location.
- You can define a `union` with various members, but only one member can hold a value at any given time.
- Union provide an efficient way of using the same memory location for multiple-purpose.

Syntax to Define and Access Union

- Declaration of union must start with the keyword `union` followed by the union name and union's member variables are declared within braces.

Syntax

```
union union_name
{
    member1_declaration;
    member2_declaration;
. . .
    memberN_declaration;
};
```

union_name is name of custom type.

memberN_declaration is individual member declaration.

- Accessing the union members:
  - You need to create an object of union to access its members.
  - Object is a variable of type union. Union members are accessed using the dot operator(.) between union's object and union's member name.

Syntax

```
union union_name union_variable;
```

Example to Define Union

Example

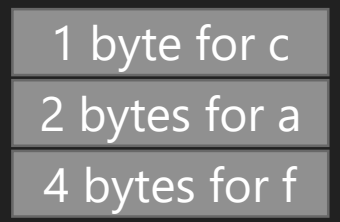
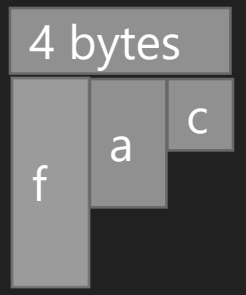
```
1  union student
2  {
3      char name[30]; // Student Name
4      int roll_no; // Student Roll No
5      float CPI; // Student CPI
6      int backlog; // Student Backlog
7  } student1;
```

- You must terminate union definition with semicolon ;.
- You cannot assign value to members inside the union definition, it will cause compilation error.

Example

```
1  union student
2  {
3      char name[30] = “ABC”; // Student Name
4  . . .
5  } student1;
```

Structure Vs. Union

COMPARISON	STRUCTURE	UNION
Basic	The separate memory location is allotted to each member of the structure.	All members of the 'union' share the same memory location.
keyword	'struct'	'union'
Size	Size of Structure = sum of size of all the data members.	Size of Union = size of the largest member.
Store Value	Stores distinct values for all the members.	Stores same value for all the members.
At a Time	A structure stores multiple values, of the different members, of the structure.	A union stores a single value at a time for all members.
Declaration	`struct ss { int a; float f; char c };` 	`union uu { int a; float f; char c };` 

Where Union should be used?

- Mouse Programming
- Embedded Programming
- Low Level System Programming

Thank you

Programming for Problem Solving (PPS)

USING

{C}
Programming

Dynamic Memory Allocation

Dynamic Memory Allocation (DMA)

- If memory is allocated at runtime (during execution of program) then it is called dynamic memory.
- It allocates memory from **heap** (*heap*: it is an empty area in memory)
- Memory can be accessed only through a pointer.

When DMA is needed?

- It is used when number of variables are not known in advance or large in size.
- Memory can be allocated at any time and can be released at any time during runtime.

malloc() function

- `malloc ()` is used to allocate a fixed amount of memory during the execution of a program.
- `malloc ()` allocates `size_in_bytes` of memory from heap, if the allocation succeeds, a pointer to the block of memory is returned else `NULL` is returned.
- Allocated memory space may not be contiguous.
- Each block contains a `size`, a pointer to the next block, and the space itself.
- The blocks are kept in ascending order of storage address, and the last block points to the first.
- The memory is not initialized.

Syntax	Description
`ptr_var = (cast_type *) malloc (size_in_bytes);`	This statement returns a pointer to `size_in_bytes` of uninitialized storage, or `NULL` if the request cannot be satisfied. **Example**: `fp = (int *)malloc(sizeof(int) *20);`

Write a C program to allocate memory using malloc.

Program

```c
#include <stdio.h>
void main()
{
    int *fp; //fp is a pointer variable
    fp = (int *)malloc(sizeof(int)); //returns a pointer to int size storage
    *fp = 25; //store 25 in the address pointed by fp
    printf("%d", *fp); //print the value of fp, i.e. 25
    free(fp); //free up the space pointed to by fp
}
```

Output

```
25
```

calloc() function

- `calloc()` is used to allocate a block of memory during the execution of a program
- `calloc()` allocates a region of memory to hold `no_of_blocks` of `size_of_block` each, if the allocation succeeds then a pointer to the block of memory is returned else `NULL` is returned.
- The memory is initialized to ZERO.

Syntax	Description

`ptr_var = (cast_type *) calloc (no_of_blocks, size_of_block);`	This statement returns a pointer to `no_of_blocks` of size `size_of_blocks`, it returns `NULL` if the request cannot be satisfied.
	
	Example:
	`int n = 20;`
	`fp = (int *)calloc(n, sizeof(int));`

Write a C program to allocate memory using calloc.

Program

```c
#include <stdio.h>
void main()
{
    int i, n; //i, n are integer variables
    int *fp; //fp is a pointer variable
    printf("Enter how many numbers: ");
    scanf("%d", &n);
    fp = (int *)calloc(n, sizeof(int)); //calloc returns a pointer to n blocks
    for(i = 0; i < n; i++)  //loop through until all the blocks are read
    {
        scanf("%d",fp); //read and store into location where fp points
        fp++; //increment the pointer variable
    }
    free(fp); //frees the space pointed to by fp
}
```

realloc() function

- `realloc()` changes the size of the object pointed to by pointer fp to specified size.
- The contents will be unchanged up to the minimum of the old and new sizes.
- If the new size is larger, the new space will be uninitialized.
- `realloc()` returns a pointer to the new space, or `NULL` if the request cannot be satisfied, in which case `*fp` is unchanged.

Syntax	Description
`ptr_var = (cast_type *) realloc (void *fp, size_t);`	This statement returns a pointer to new space, or NULL if the request cannot be satisfied. **Example**: `fp = (int *)realloc(fp,sizeof(int)*20);`

Write a C program to allocate memory using realloc.

Program

```c
#include <stdio.h>
void main()
{
    int *fp; //fp is a file pointer
    fp = (int *)malloc(sizeof(int)); //malloc returns a pointer to int size storage
    *fp = 25; //store 25 in the address pointed by fp
    fp =(int *)realloc(fp, 2*sizeof(int)); //returns a pointer to new space
    printf("%d", *fp); //print the value of fp
    free(fp); //free up the space pointed to by fp
}
```

Output

```
25
```

free() function

- `free` deallocates the space pointed to by fp.
- It does nothing if fp is `NULL`.
- fp must be a pointer to space previously allocated by `calloc`, `malloc` or `realloc`.

Syntax	Description
`void free(void *);`	This statement free up the memory not needed anymore. **Example**: `free(fp);`

Write a C program to sort numbers using malloc

Program

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i,j,t,n;
    int *p;
    printf("Enter value of n: ");
    scanf("%d", &n);
    p=(int *) malloc(n * sizeof(int));
    printf("Enter values\n");
    for(i=0; i<n; i++)
        scanf("%d", &p[i]);
    for(i=0; i<n; i++)
    {
        for(j= i+1; j<n; j++)
        {
```

Program (cont.)

```c
            if(p[i] > p[j])
            {
                t = p[i];
                p[i] = p[j];
                p[j] = t;
            }
        }
    }
    printf("Ascending order\n");
    for(i=0; i<n; i++)
        printf("%d\n", p[i]);
    free(p);
}
```

Write a C program to find square of numbers using calloc

Program

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i,n;
    int *p;
    printf("Enter value of n: ");
    scanf("%d",&n);
    p=(int*)calloc(n,sizeof(int));
    printf("Enter values\n");
    for(i=0;i<n;i++)
        scanf("%d",&p[i]);
    for(i=0;i<n;i++)
        printf("Square of %d = %d\n", p[i],
 p[i] * p[i]);
    free(p);
}
```

Output

```
Enter value of n: 3
Enter values
3
2
5
Square of 3 = 9
Square of 2 = 4
Square of 5 = 25
```

Write a C program to add/remove item from a list using realloc

Program

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i, n1, n2;
    int *fp;
    printf("Enter size of list: ");
    scanf("%d", &n1);
    fp=(int *) malloc (n1 * sizeof(int));

    printf("Enter %d numbers\n", n1);
    for(i = 0; i < n1; i++)
        scanf("%d", &fp[i]);

    printf("The numbers in the list are\n");
    for(i = 0; i < n1; i++)
        printf("%d\n", fp[i]);
```

Program (cont.)

```c

    printf("Enter new size of list: ");
    scanf("%d", &n2);

    fp = realloc(fp, n2 * sizeof(int));
    if(n2 > n1)
    {
        printf("Enter %d numbers\n", n2 - n1);
        for(i = n1; i < n2; i++)
            scanf("%d", &fp[i]);
    }
    printf("The numbers in the list are\n");
    for(i = 0; i < n2; i++)
        printf("%d\n", fp[i]);
}
```

Practice Programs

1) Write a C program to calculate sum of n numbers entered by user.
2) Write a C program to input and print text using DMA
3) Write a C program to read and print student details using structure and DMA

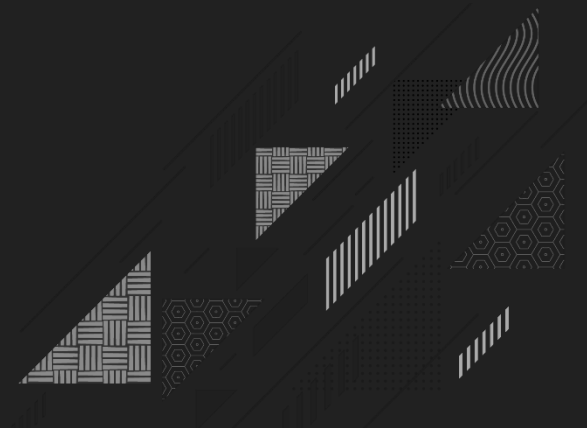
Thank you

USING

{C}
Programming

File Management

File management is what you have, and how you want to manipulate it. - Anonymous



Why File Management?

- In real life, we want to store data permanently so that later we can retrieve it and reuse it.
- A file is a collection of characters stored on a secondary storage device like hard disk, or pen drive.
- There are two kinds of files that programmer deals with:
  - Text Files are human readable and it is a stream of plain English characters
  - Binary Files are computer readable, and it is a stream of processed characters and ASCII symbols

Text File

Hello, this is a text file. Whatever written here can be read easily without the help of a computer.

Binary File

```
11010011010100010110111010
10111010111010011010100010
11011101010111010111010011
```

File Opening Modes

- We can perform different operations on a file based on the file opening modes

Mode	Description
r	Open the file for reading only. If it exists, then the file is opened with the current contents; otherwise an error occurs.
w	Open the file for writing only. A file with specified name is created if the file does not exists. The contents are deleted, if the file already exists.
a	Open the file for appending (or adding data at the end of file) data to it. The file is opened with the current contents safe. A file with the specified name is created if the file does not exists.
r+	The existing file is opened to the beginning for both reading and writing.
w+	Same as w except both for reading and writing.
a+	Same as a except both for reading and writing.

Note: The main difference is w+ truncate the file to zero length if it exists or create a new file if it doesn't. While r+ neither deletes the content nor create a new file if it doesn't exist.

File Handling Functions

- Basic file operation performed on a file are opening, reading, writing, and closing a file.

Syntax	Description
`fp=fopen(file_name, mode);`	This statement opens the file and assigns an identifier to the `FILE` type pointer `fp`. Example: `fp = fopen("printfile.c","r");`
`fclose(filepointer);`	Closes a file and release the pointer. Example: `fclose(fp);`
`fprintf(fp, "control string", list);`	Here `fp` is a file pointer associated with a file. The control string contains items to be printed. The list may includes variables, constants and strings. Example: `fprintf(fp, "%s %d %c", name, age, gender);`

File Handling Functions

Syntax	Description
`fscanf(fp, “control string”, list);`	Here `fp` is a file pointer associated with a file. The control string contains items to be printed. The list may includes variables, constants and strings. Example: `fscanf(fp, "%s %d", &item, &qty);`
`int getc( FILE *fp);`	`getc()` returns the next character from a file referred by `fp`; it require the `FILE` pointer to tell from which file. It returns `EOF` for end of file or error. Example: `c = getc(fp);`
`int putc(int c, FILE *fp);`	`putc()` writes or appends the character `c` to the `FILE` `fp`. If a `putc` function is successful, it returns the character written, `EOF` if an error occurs. Example: `putc(c, fp);`

File Handling Functions

Syntax	Description
`int getw( FILE *pvar);`	`getw()` reads an integer value from `FILE` pointer `fp` and returns an `integer`. Example: `i = getw(fp);`
`putw(int, FILE *fp);`	putw writes an integer value read from terminal and are written to the `FILE` using `fp`. Example: `putw(i, fp);`
`EOF`	`EOF` stands for “End of File”. `EOF` is an integer defined in `<stdio.h>` Example: `while(ch != EOF)`

Write a C program to display content of a given file.

Program

```c
#include <stdio.h>
void main()
{
    FILE *fp; //p is a FILE type pointer
    char ch; //ch is used to store single character
    fp = fopen("file1.c","r"); //open file in read mode and store file pointer in p
    do {  //repeat step 9 and 10 until EOF is reached
        ch = getc(fp); //get character pointed by p into ch
        putchar(ch); //print ch value on monitor
    }while(ch != EOF); //condition to check EOF is reached or not
    fclose(fp); //free up the file pointer pointed by fp
}

```

Write a C program to copy a given file.

Program

```c
#include <stdio.h>
void main()
{
    FILE *fp1, *fp2; //p and q is a FILE type pointer
    char ch; //ch is used to store temporary data
    fp1 = fopen("file1.c","r"); //open file “file1.c” in read mode
    fp2 = fopen("file2.c","w"); //open file “file2.c” in write mode
    do { //repeat step 9 and 10 until EOF is reached
        ch = getc(fp1); //get character pointed by p into ch
        putc(ch, fp2); //print ch value into file, pointed by pointer q
    }while(ch != EOF); //condition to check EOF is reached or not
    fclose(fp1); //free up the file pointer p
    fclose(fp2); //free up the file pointer q
    printf("File copied successfully...");
}
```

File Positioning Functions

- `fseek`, `ftell`, and `rewind` functions will set the file pointer to new location.
- A subsequent read or write will access data from the new position.

Syntax	Description
`fseek(FILE *fp, long offset, int position);`	`fseek()` function is used to move the file position to a desired location within the file. **`fp`** is a `FILE` pointer, **`offset`** is a value of datatype `long`, and **`position`** is an `integer` number. **Example**: /* Go to the end of the file, past the last character of the file */ `fseek(fp, 0L, 2);`
`long ftell(FILE *fp);`	`ftell` takes a file pointer and returns a number of datatype `long`, that corresponds to the current position. This function is useful in saving the current position of a file. **Example**: /* n would give the relative offset of the current position. */ `n = ftell(fp);`

File Positioning Functions

Syntax	Description
`rewind(fp);`	`rewind()` takes a file pointer and resets the position to the start of the file. **Example**: /* The statement would assign 0 to n because the file position has been set to the start of the file by rewind. */ `rewind(fp);`

Write a C program to count lines, words, tabs, and characters

Program

```
#include <stdio.h>
void main()
{
    FILE *p;
    char ch;
    int ln=0,t=0,w=0,c=0;
    p = fopen("text1.txt","r");
    ch = getc(p);
    while (ch != EOF) {
        if (ch == '\n')
            ln++;
        else if(ch == '\t')
            t++;
        else if(ch == ' ')
            w++;
        else
```

Program (contd.)

```
            c++;

        ch = getc(p);
    }
    fclose(p);
    printf("Lines = %d, tabs = %d, words = %d, characters = %d\n",ln, t, w, c);
}
```

Output

```
Lines = 22, tabs = 0, words = 152, characters = 283
```

Practice Programs

1) Write a C program to write a string in file.
2) A file named data contains series of integer numbers. Write a C program to read all numbers from file and then write all the odd numbers into file named “odd” and write all even numbers into file named “even”. Display all the contents of these file on screen.
3) Write a C program to read name and marks of n number of students and store them in a file.
4) Write a C program to print contents in reverse order of a file.
5) Write a C program to compare contents of two files.
6) Write a C program to copy number of bytes from a specific offset to another file.
7) Write a C program to convert all characters in UPPER CASE of a File.

Thank you