C programming

Page No.
Date: / /

gcc HelloWorld.C gcc → C compiles → provoking the compiles

./a.exe

./a.out (in linux, mac)

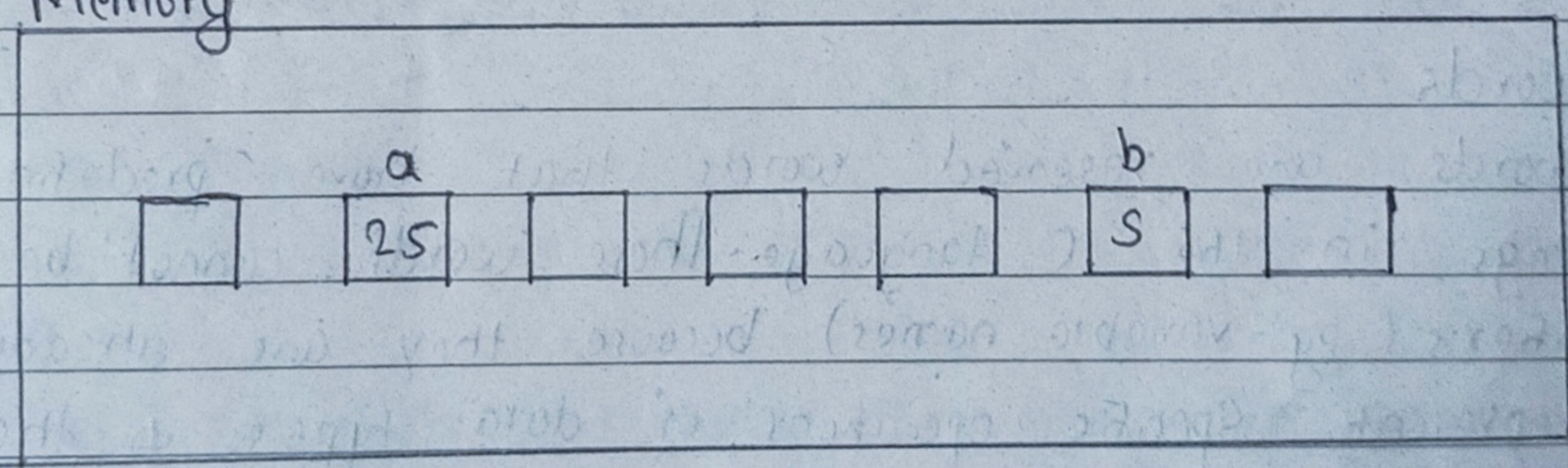
chapter - 1

- Variables
- Keywords
- Constants
- Comments
- Structure
- Compilation

- Variables

Variable is the name of a memory location which stores some data.

Memory



Rules

a. Variables are case sensitive
b. 1st character is alphabet or '_' underscore
c. no comma / ~~bac~~ blank space
d. No other symbol other than '_' underscore

Page No.
Date: / /

- Rules to declare the variables

- They must begin with letter or underscore (_)
- First character may be followed by letters or digits.
- Both lowercase and uppercase letters are distinct.
  Example: Total and total are not same
- It should not be a keyword
- white space, Dollar sign are not allowed.
  Example: John, delhi, first_tag, int_type -> (Valid)
  Price$, char, group one, 123, (area) -> (invalid)

- C Tokens

In C programming, tokens are the smallest individual units of a program. These tokens are used to build the syntax of the language. The various types of tokens in C are as follows.

1) Keywords
Keywords are reserved words that have predefined meanings in the C language. These words cannot be used as identifiers (eg:- variable names) because they are already used to represent specific operations or data types in the language: Example include 'int', 'float', if, while, return, etc.

2) Identifiers
Identifiers are user-defined names given to variables, functions, arrays, etc. in a C program. An identifier can consist of letters, digits and underscores, but it must start with a

letter or an underscore. Identifiers are case-sensitive. For example, 'sum', ~~total~~ 'total_amount', 'calculateArea', etc.

3) Constants
Constants are fixed values that do not change during program execution. There are several types of constants.

- Integer Constants: Whole numbers without any fractional part. For example, '10', '-5', 0XFF (hexadecimal), etc.
- Floating point Constants: Numbers with a fractional part or exponent notation. For example, '3.14', 2.0e3, -0.5, etc.
- Character Constants: Single characters enclosed in single quotes. For example, 'A', '5', '\n' (newline character), etc.
- String Constants: Sequence of characters enclosed in double quotes. For example, "Hello", "12.34", "C programming", etc.

4) String literals:
- These are sequences of characters enclosed in double quotes.
- Used to represent text or strings of characters.
- Example: "Hello, world!"

5) Operators
Operators are symbols used to perform operations on operands. C provides a wide range of operators for arithmetic, relational, logical, bitwise, etc. operations. Examples

include +, -, *, /, % (assignment operators), &&, ||, ! (logical operator), etc.

6) Special Symbols
Special characters used for specific purposes in the syntax of the language.
Examples: '#' (hash): used for preprocessor directives.
'?:' (ternary conditional operator): Used for conditional expressions, braces {}, parentheses (), square bracket [], comma, semicolon, etc.

7) Comments
Comments are text in the source code that is ignored by the compiler. They are used to add explanations or notes to the code. C supports two types of comments.

- Single line comments: Begin with "//" and extend to the end of the line.
- Multi line comments: Enclosed between '/*' and '*/', and can span multiple lines.

8) Delimiters
- Special symbols used to mark the beginning or end of a block of code, or to separate items in a list.
- Examples: '{', '}', '(', ')', '[', ']', ',', ';'.

Page No.
Date: / /

- Escape Sequence
- An escape sequence in C language is a sequence of character that doesn't represent itself when used inside string literal or characters.

- It is composed of two or more characters starting with backslash \. For example: \n represents new line.

\t represents tab. (Horizontal)
\\ ~~repr~~ represents Backslash
\' represents Single Quote
\" represents Double Quote
\v represents vertical tab

- Variables
- Unlike Constants that remain unchanged during the execution of the program, a variable may take different value at different time.

- Basic structure of a C program

1. Documentation section
2. Preprocessor section
3. Definition section
4. Global declaration section
5. main function
6. Sub function

1. Documentation

- This section provides a description of the program, including its name, programmes's name, creation date, and time.

- It is specified at the starting of the program using comments.

- Examples:

```
// Program to calculate the sum of two numbers
// Author: Bibek Shah
// Date: March 16, 2024
```

2. Preprocessor section

- Here, we declare all the necessary header files using #include directives.

- Header files allow us to access predefined functions and features.

- Example:

```
#include <stdio.h>
#include <math.h>
```

3. Definition

- Preprocessor directives (starting with #) are used in this section.

Page No.
Date: / /

- The #define preprocessor creates constants throughout the program.

- Example:
  #define PI 3.14159

4. Global Declaration
- This section contains global variables, function declarations, and static variables.

- Variables and functions declared here can be used anywhere in the program.

- Example:
  int num = 18;

5. Main() function
- Every C program must have a main() function.

- Inside the curly braces of main(), we perform declarations and execute operations.

- The return type can be int or void.

- void main() indicates that the program won't return any value

- int main() indicates that the program will return an integer value.

- Example:

```
int main() {
    // Code goes here
    return 0;
}
```

6. Sub Programs:

- Uses-defined functions are called in this section.

- Control shifts to these functions when called from main() or elsewhere.

- Example:

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

```
/* program demonstrating basic structure of C program */
#include <stdio.h>
# define PI 3.14

// function declaration / prototype
float area (float r);

int main() {
  float a, r;
  printf("Enter radius of circle: \n");
```

```
scanf("%f", &r);
a = area (r);
printf("The area of circle is %f", a);
return 0;
}

// function definition
float area (float r)
{
  float ar;
  ar = PI *r*r;
  return ar;
}
```

- Primary Data Types

Integer			Character	
Signed	Unsigned			Size in bytes ~~all takes~~
2 byte → int	unsigned int	→ 2 byte	char	1 bytes
2 byte → short int	unsigned short int	→ 2 byte	signed char	
4 byte → long int	unsigned long int	→ 4 byte	unsigned char	

Floating Point Type			Void
float	double	long double	
4 byte	8 byte	10 byte	

- boolean } These data type does not exists in C.
  String

- C introduced in 1970s

- Some main data types are

```
int age = 22;
float pi = 3.14;
char hashtag = '#';
```

- Constants
  Values that don't change (fixed)

Types

Integer Constants	Floating point Constants	Character Constants
1, 2, 3, 0, -1, -2	1.0, 2.0, 3.14,	'a', 'b', 'A', '#', '8' '*', '@',

- keywords
  Reserved words that have special meaning to the compiler
  ↓
  32 keywords in C

Page No.
Date : / /

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

- Program structure

```
#include <stdio.h>
int main() {
    printf("Hello world");
    return 0;
}
```

preprocessor directives

- Comments
lines that are not part of program

Single line
//

multiline
/*
*/

- Output

Cases

1. Integers
printf("age is %d", age);

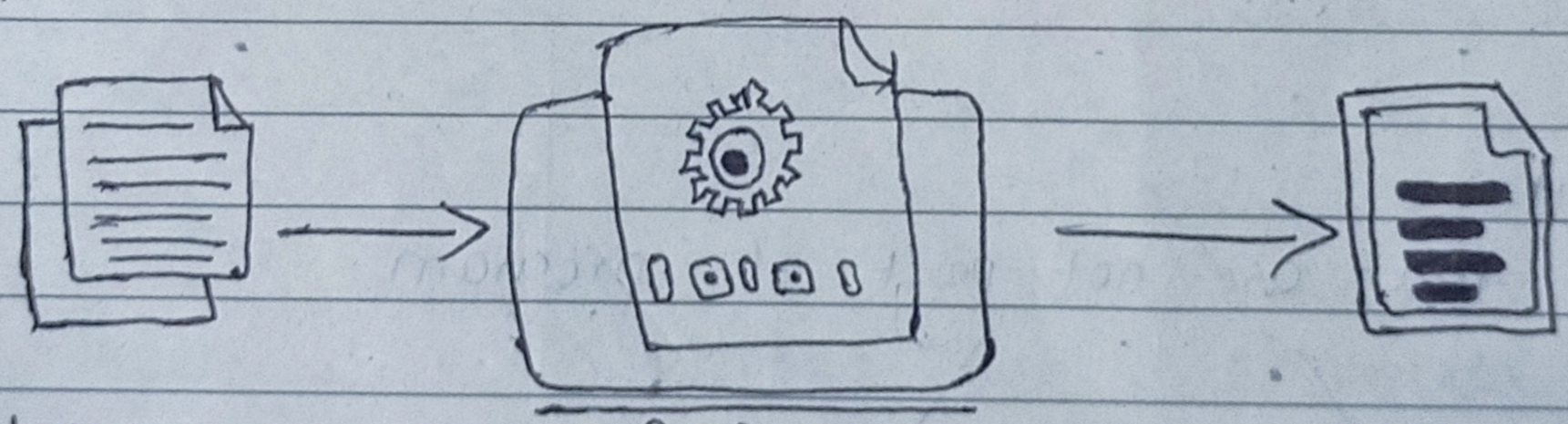
Format Specifiers

2. ~~real~~ floating numbers
printf("value of pi is %f", pi);

3. characters
printf("star looks like this %c", star);

- Input
scanf("%d", &age);

- Compilation
A computer program that translates C code into machine code.



Practise Qs. 1

1. Write a program to calculate area of a square. (side is given)
2. Write a program to calculate area of a circle. (~~side~~ radius is given)

Page No.
Date: / /

Homework set

a. Write a program to calculate perimeter of rectangle. Take sides, a & b, from the user.

b. Take a number (n) from user & output its cube (n*n*n).

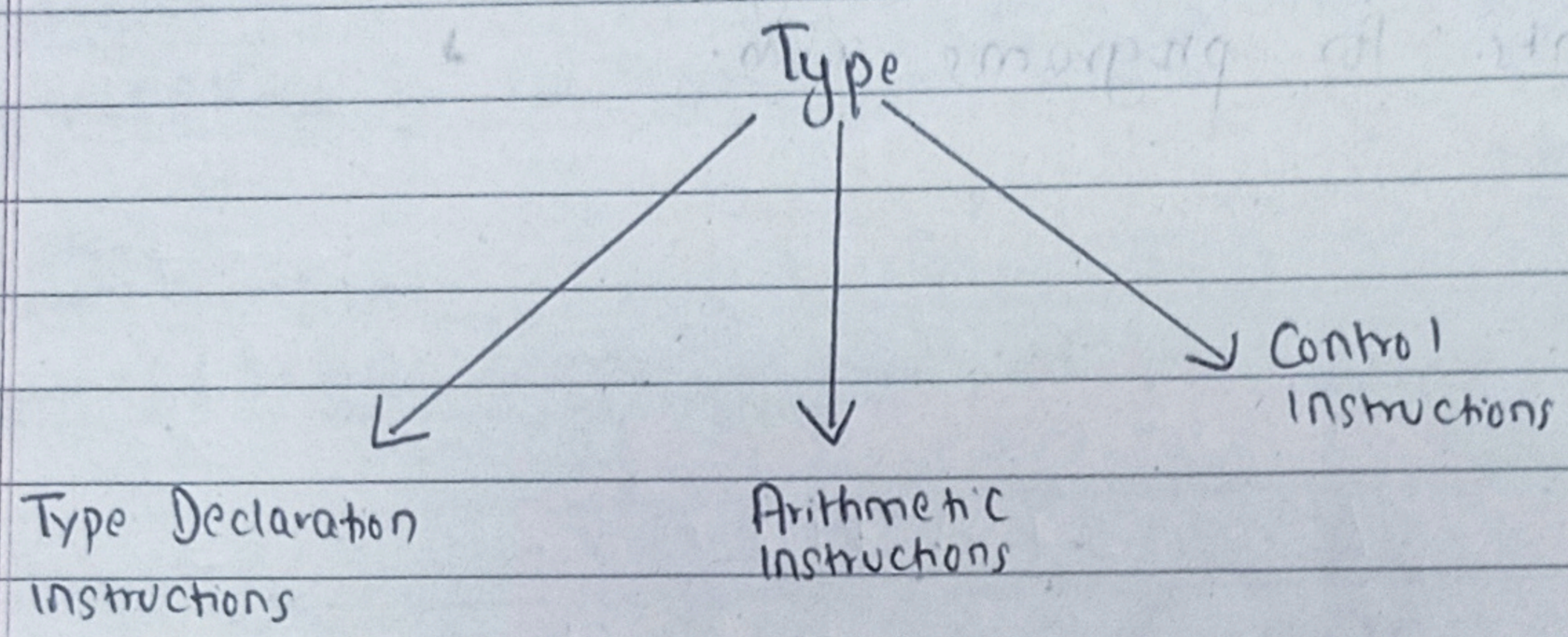
c. Write comments for programs a & b.

Chapter 2
Instruction & operators

Page No.
Date: / /

- Instructions

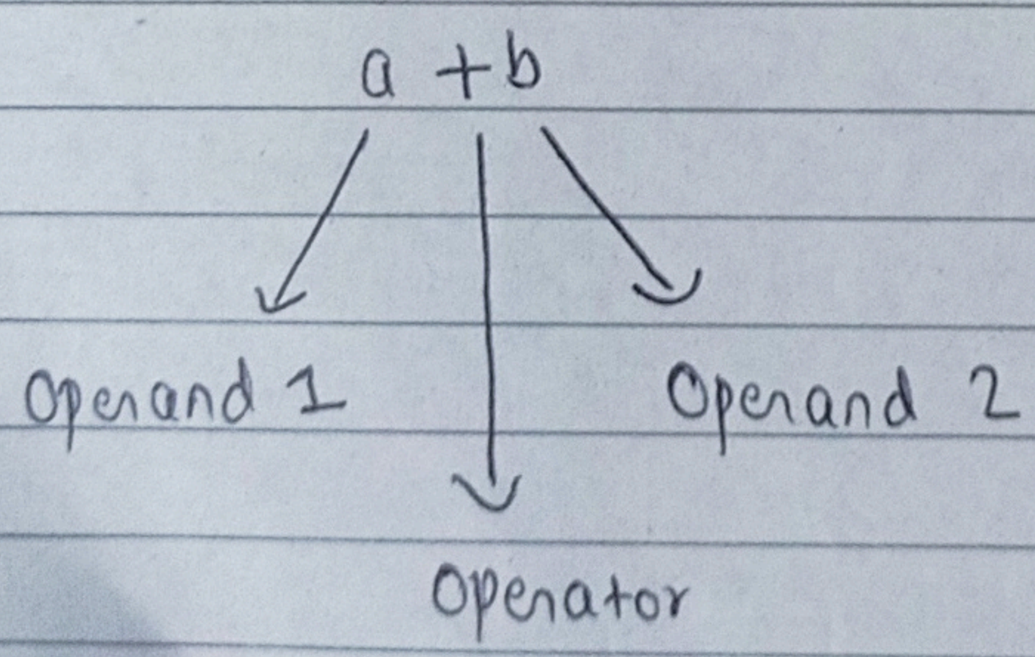
These are statements in a Program



- Type Declaration Instructions → Declare var before using it

```
int a,b,c;
a=b=c=1;
```

- Arithmetic Instructions



Note - Single variable on the LHS

VALID	INVALID
a=b+c	b+c=a

a = b * c a = bc
a = b / c a = b ^ c

Note: pow(x, y) for x to the power y

#include <math.h>

- Modular Operator
Returns remainder for int

3 % 2 = 1
-3 % 2 = -1

- Type Conversion

int op int -> int
int op float -> float
float op float -> float

Solve:
int a = 1999999;
int a = (int) 1.999999;

Conversion
implicit explicit

Page No.
Date : / /

- Operator Precedence

*, /, % $x = 4 + 9 * 10$
↓ $= 4 + 90$
+, - $= 94$
↓
= $x = 4 * 3 / 6 * 2 = 4$
→ assignment operator

Associativity (for same precedence)
left to Right
$x = 4 * 3 / 6 * 2$
$= 12 / 6 * 2$
$= 2 * 2$
$= 4$

- Practice Qs

a. $5 * 2 - 2 * 3$
$= 10 - 6$
$= 4$

b. $5 * 2 / 2 * 3$
$10 / 2 * 3$
$5 * 3$
15

c. $5 * (2 / 2) * 3$
$5 * 1 * 3$
15

d. $5 + 2 / 2 * 3$ → Same precedence left to right
$= 5 + \cancel{2/6}\ 1 * 3$
$= 5 + 3$
$= 8$

; terminator

- Control Instructions
  Used to determine flow of program

a. Sequence Control
b. Decision Control -> if-else
c. Loop control -> for, while, do-while
d. Case control ->

- Operators

a. Arithmetic Operators -> +, -, *, /, %
b. Relational " -> ==, >, >=, <, <=, !=
c. Logical " -> &&, ||, !
d. Bitwise " ->
e. Assignment " -> =, +=, -=, *=, /=, %=
f. Ternary " ->

- Operator Precendence

Priority	Operator
1	! (logical)
2	*, /, % (Arithmetic)
3	+, - (Arithmetic)
4	<, <=, >, >= (Relational)
5	==, != (Relational)
6	&& (logical)
7	|| (logical)
8	= (Assignment)

- Assignment Operators ← short hand operators

  a = a + b

  a += b

- Practice Qs.

a. Write a program to check if a number is divisible by 2 or not.

b. Write a program to check if a number is odd or even.

c. C program to print the average of 3 numbers.

d. Write a program to check if given character is digit or no

e. Program to print the smallest number.

```
printf("%d", number%2==0);
```

→ #incude <ctype.h>

isdigit(character)

Chapter-3

Conditional Statements

Page No.
Date: / /

- Operator

An operator is a symbol that tells the Computer to perform certain manipulations.

- Operator Precedence

- Operator precedence determines which operator is performed first in an expression with more than one operators with different precedence.

For example: Solve

lower precedence
↓
10 + 20 * 30
↑ upper precedence

So, 10 + 600
= 610

- Operators Associativity

- Operators Associativity is used when two operators of same precedence appear in an expression. Associativity can be either left to Right or Right to Left.

- For example: '*' and '/' have same precedence and their associativity is Left to Right, so the expression

"100/10 *10" is treated as "(100/10) *10".

- left Associativity

3*8/4%4*5

- Right Associativity

a+=b*=c--=5

a+= b*= c--=5

Examples

10-3%8+6/4

17-8/4*2+3--+5

10-3+6/4

10-3+1

7+1

8

17-8/4*2+3-8%6

17-2*2+3-8%6

17-4+3-8%6

13+3-8%6

16-8%6

10

Chapter-3

Conditional Statements

Page No.
Date: / /

- Types
  - if-else
  - Switch

- if-else

Syntax

```
if (Condition){
    //do something if TRUE
}
else{
    //do something if FALSE
}
```

- else if

```
if (condition 1){
    //do something if TRUE.
}
else if (Condition 2){
    //do something if 1st is FALSE & 2nd is TRUE
}
```

- Ternary Operator (Conditional operator)

```
Condition? doSomething if TRUE : doSomething if FALSE;
```

Date: / /

Conditional Operators

- Switch

```
switch(number){
case C1: //do something
        break;
case C2: //do something
        break;
default: //do something
}
```

- Switch Properties

a. Cases can be in any order
b. Nested switch (switch inside switch) are allowed.

- Practice Qs

a. Write a program to check if a student passed or failed.
marks > 30 is PASS
marks <= 30 is FAIL

b. Write a program to give grades to a student
marks < 30 is C
30 <= Marks < 70 is B
70 <= marks < 90 is A
90 <= marks <= 100 is A+

c. Write a program to find if a character entered by user is uppercase or not.

Page No.
Date: / /

ASCII Code

A - Z
65 - 90

a - z
97 - 122

Q. d. Write a program to check if a given number is Armstrong number or not.
(Search what is Armstrong number)

e. Write a program to check if the given number is a natural number.
(Natural numbers start from 1)

Chapter 4

loop Control Instruction

Page No.
Date: / /

- loop Control Instructions
To repeat some parts of the program

Types
- for
- while
- do while

- for Loop

```
for(initialisation; condition; updation){
    //do something
}
```

- Print 1 to 100

i means iterator or counter

Q. Print the Numbers from 0 to 10
0 1 2 3 4 5 6 7 8 9 10

- Increment Operator
  i++ (post increment operator) //use, then increase
  ++i (pre increment operator) //increase, then use

- decrement operator
  i-- (post decrement)
  --i (pre decrement)

Page No.
Date: / /

- Special Things

- Increment Operator
- Decrement Operator
- loop counter can be float or even character
- Infinite Loop

- While loop

```
while (condition){
  //do something
}
```

Q Print the numbers from 0 to n, if n is given by user
n=4

- do while Loop

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

Q. Print the Sum of first n Natural Numbers.
Also, print them in reverse.

Q. Print the table of a number input by the user.
n=2

- break statement

↓

exit the loop

Q. Keep taking numbers as input from user until user enters an odd number.

Q. Keep taking numbers as input from user until user enters a number which is multiple of 7.

- Continue statement

↓

skip to next iteration

Q. Print all numbers from 1 to 10 except for 6.

Q. Print all the odd numbers from 5 to 50.

Q. Print the factorial of a number n.

Q. Print reverse of the table for a number n.

Q. Calculate the sum of all number between 5 and 50. (including 5 & 50).

Page No.
Date : / /

Homework Set

a. Search on what a "nested loop" is & print this pattern using it.

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

b. Write a program to check if a number is prime or not.

c. Write a program to print prime numbers in a range.

Chapter-5

Functions and Recursion

- Functions

↓

block of code that performs particular task

Take Argument → [Do Work] → Return Result

it can be used multiple times
increase code reusability

- Syntax 1

Function Prototype / Declaration

```
void printHello();
```

Syntax 2

Function Definition

```
void printHello()
    printf("Hello");
}
```

Syntax 3

Function Call

Page No.
Date: / /

```
int main() {
    printHello();  ←
    return 0;
}
```

Q. Write 2 functions - one to print "Hello" & second to print "good bye".

Q. Write a function that prints Namaste if user is ~~indian~~ Nepali & Bonjour if the user is french.

- Properties

- Execution always starts from main.
- A function gets called directly or indirectly from main.
- There can be multiple functions in a program.

- Function Types

Library function
Special functions inbuilt in C

scanf(), printf()

User-defined
declared & defined by programmer

- Passing ~~value~~ Arguments
functions can take value & give some value

parameter

return value

- void printHello();

parameters

void printTable(int n);

int sum(int a, int b);

return type

parameters

sum=a+b
↓
return

- Argument v/s Parameters

a)	values that are passed in function call	values in function declaration & definition
b)	Used to send value	Used to receive value
c)	actual parameters	formal parameters

Note:

a. function can only return one value at a time
b. Changes to parameters in function don't change the

values in calling function.
Because a copy of argument is passed to the function.

Q. Use library functions to calculate the square of a number given by user.

↓

number x number

<math.h> -> pow(number, 2) => number^2

pow(n, 4) => n^4

Q. Write functions to calculate area of square, a circle & a rectangle.

↓ side x side

↓ πr^2

↓ length x breadth

Date: / /

- Recursion
When a function calls itself, it's called recursion.

Q. Print "Hello World" 5 times.

4:43:00
Skipped recursion topic

Chapter - 7

Arrays

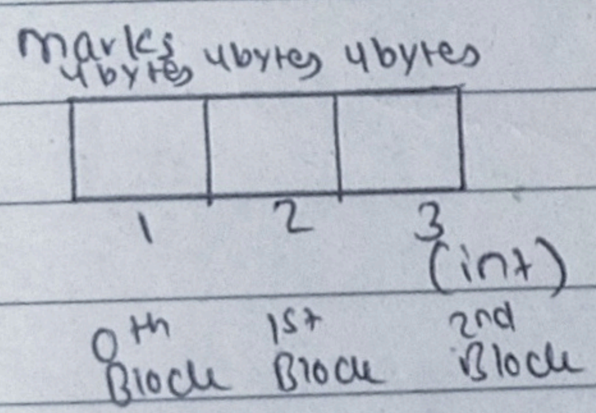
Page No.
Date : / /

- An array in C is a collection of elements of the same data type, stored in contiguous memory locations. Each element in the array is accessed by its index.

- Collection of similar data types stored at contiguous memory locations.

- Syntax

```
int marks[3];
char name[10];
float price[2];
```



0-based indexing

- Input & output

```
scanf("%d",&marks[0]);

printf("%d", marks[0]);
```

Q. Write a program to enter price of 3 items & print their final cost with gst.

3 items → float + 18% gst

- ~~Initital~~ Initialization of Array

int marks[] = {97, 98, 89};

int marks[3] = {97, 98, 89};

marks

97	98	89
0	1	2

Memory Reserved : 12 bytes.

- char name[10];

  char → 1 byte

  name[10] → 10×1 = 10 bytes

int roll[30];

int → 4 bytes

roll[30] → 30×4 = 120 bytes

- Multidimensional Arrays

2D Arrays

int arr[][] = {{1,2},{3,4}}; //Declare

//Access

arr[0][0] → 1

arr[0][1] → 2

00	01	
1	2	← 1st array
10	11	
3	4	← 2nd array

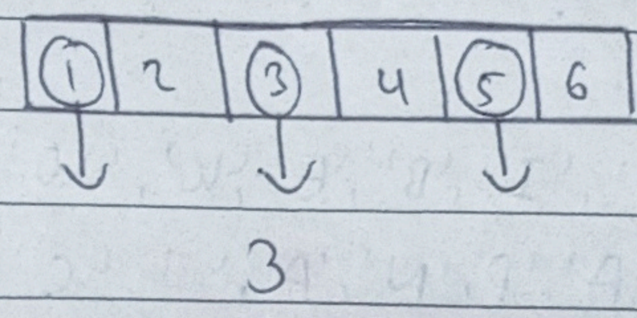
arr

Page No.
Date : / /

arr[1][0] -> 3
arr[1][1] -> 4

1	2	3	4
00	01	10	11

Q. Write a function to count the number of odd numbers in an array.



7:01:00

Page No.
Date: / /

Chapter - 8

Strings

- Strings
A character array terminated by a '\0' (null character)
null character denotes string termination.

Example

```
char name[] = {'B','I','B','E','k','\0'};
char class[] = {'A','P','N','A',' ','C','O','L','L','E','G','E','\0'}
```

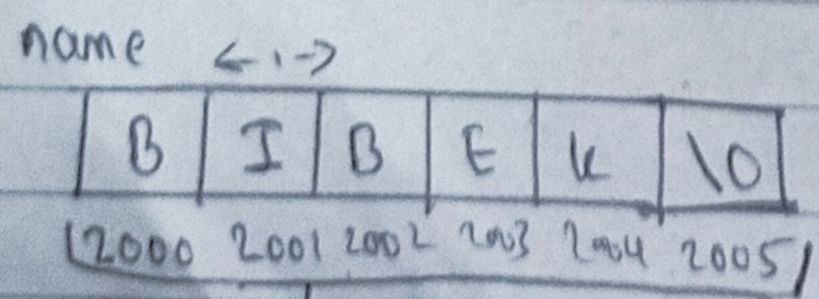
normal character array
```
char name[] = {'B','I','B','E','k'};
```

String of words
```
char name[] = {'B','I','B','E','k','\0'};
```

- Initialising strings

```
char name[] = {'B','I','B','E','k','\0'}
char name[] = "BIBEk";

char class[] = {'A','P','N','A',' ','C','O','L','L','E','G','E','\0'};
char class[] = "APNA COLLEGE";
```



address
↓
Contiguous memory address

Page No.
Date : / /

Q. Create a string firstName & lastName to store details of user & print all the characters using a loop.

- String format Specifier
  ↓
  "%s"

```
char name[] = "~~shra~~ Bibek";
printf("%s", name);
```

Q. Ask the user to enter their firstName & print it back to them.
Also try this with their fullName.

- Important

scanf() cannot input multi-word strings with spaces

Here,
gets() & puts() come into picture.
↓
fgets()

- String functions

gets(str) -> Dangerous & Outdated
input a string (even multiword)

puts(str)
output a string

Date : /

fgets(str, n, ~~file~~) → stdin (standard input)

Stops when n-1 chars input or new line is entered

7:43:00

- Common string functions (string.h)

a) strlen()
b) strcpy()
c) strcmp()
d) strcat()
e) strupr()
f) strlwr()
g) strrev()

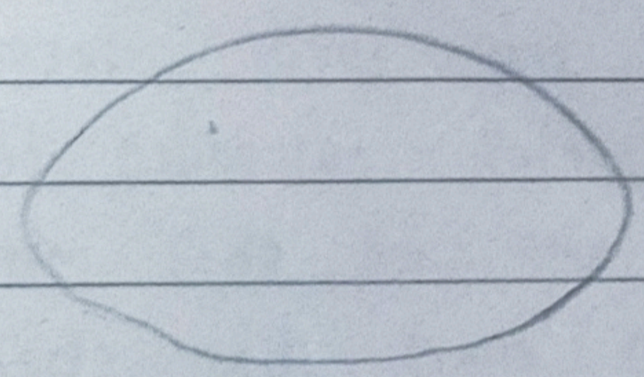
Page No.
Date : / /

- Algorithm and Flowchart

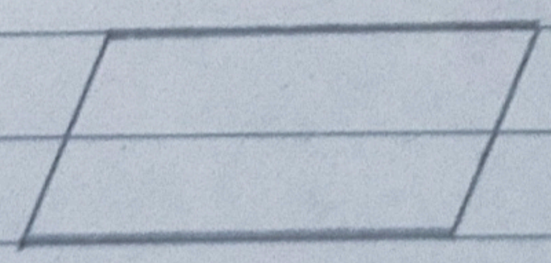
Algorithm is a complete step by step representation of the solution of the problem, represented in English like language. It's essentially a procedure or formula that describes the exact sequence of operations to be performed.

A Flowchart is a graphical representation of an algorithm. It uses different shapes to represent different types of steps or actions and arrows to show the flow or sequence of these steps.

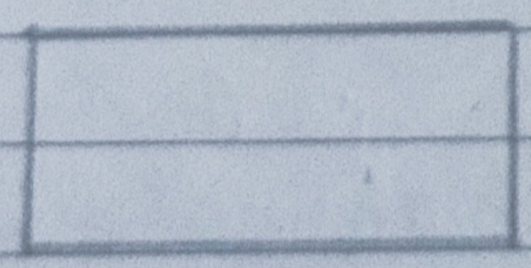
a) Start, Stop



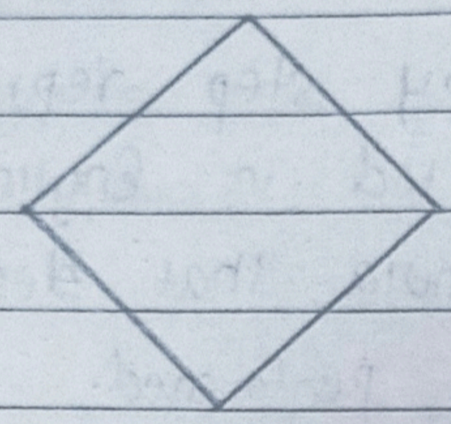
b) Read, Print



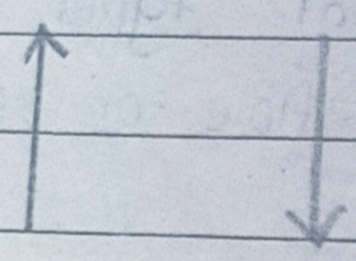
c) Processing statements



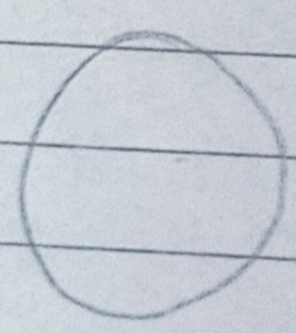
d) Condition check



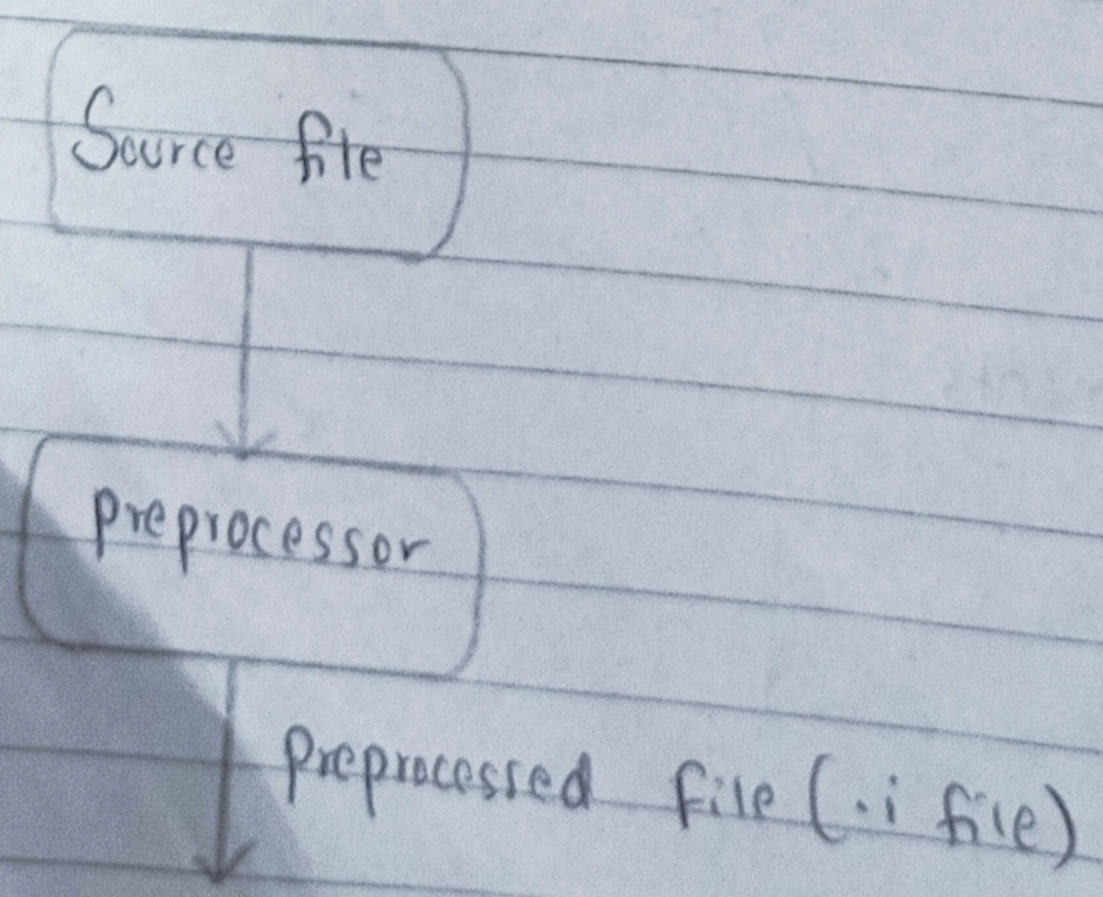
e) Direction of flow

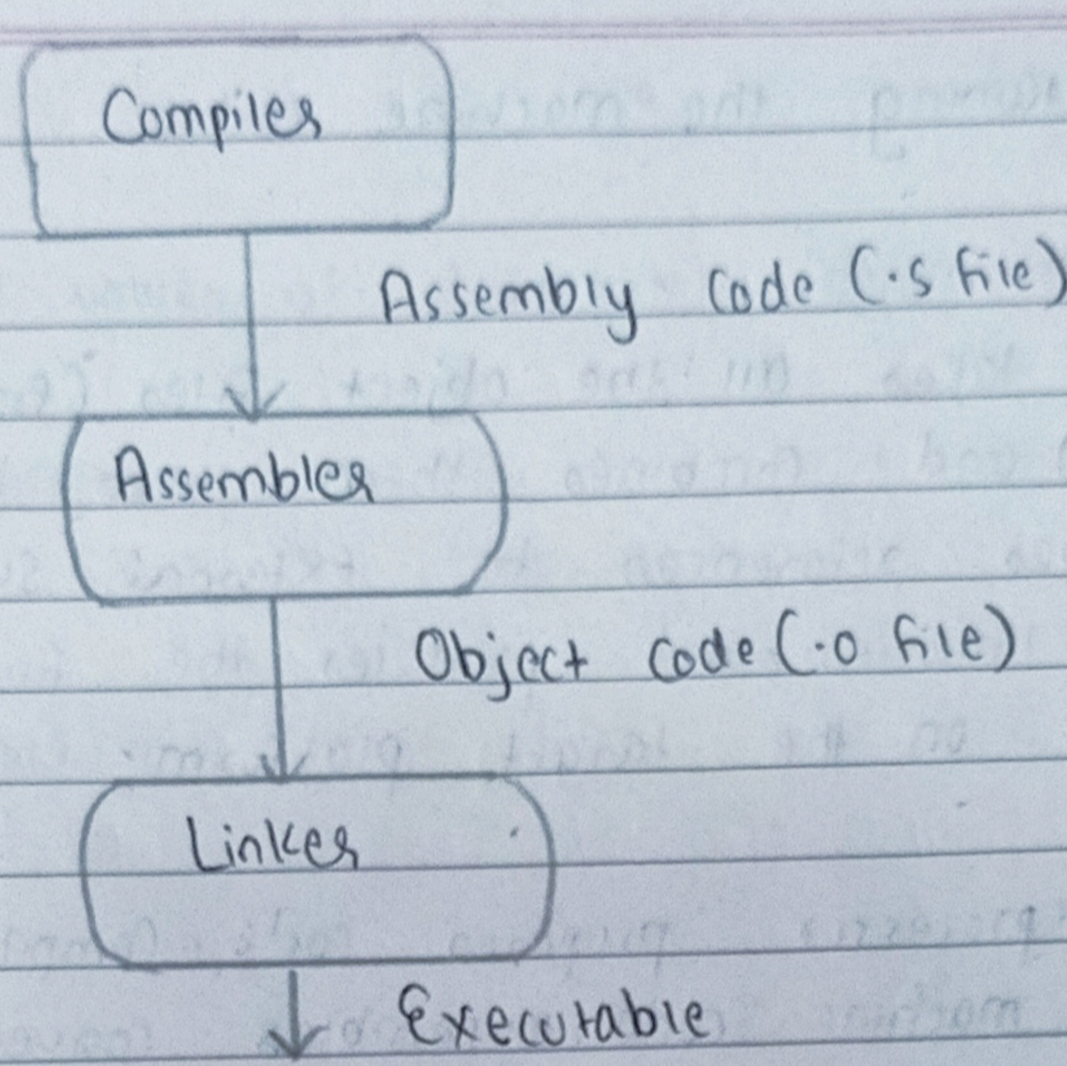


f) Connectors
(for longer flow charts)



- C compilation process





a. Preprocessor
The preprocessor takes care of directives like '#include' and '#define'. It processes these before actual Compilation, essentially preparing the code for the Compiler. It removes comments from the code and then generates code that the Compiler
b. will then work with.

b. Compiler
The Compiler translates the preprocessed C code into assembly code. It checks for Syntax errors and generates object files (.o files). Also optimizes the code (if optimizations are enabled) to improve performance.

c. Assembler
The ~~Compiler~~ assembler converts the assembly code into machine code specific to the architecture (like x86, ARM, etc) It produces object files as well. Generates object files

Page No.
Date: / /

(.o files) containing the machine code.

d. linker
The linker takes all the object files (from both your code and libraries) and combines them into a single executable file. It resolves references to external symbols, like functions in libraries, and produces the final executable that can be run on the target platform.

In short, preprocessor prepares code, compiler converts it to assembly or machine code, assembler converts assembly to machine code, and linker combines all code and libraries into an executable.

Structures

↓

a collection of values of different data ~~ds~~ types.

Example

For a student store the following:

name (string)

roll no (Integer)

gpa (Float)

Syntax

```
struct student{
    char name[100];
    int roll;
    float gpa;
};
```

```
struct student s1;
s1.gpa = 4.0;
```

Structures in Memory

```
struct student{
    char name[100];
    int roll;
    float cgpa;
};
```

name	roll	cgpa
	4 byte	
2010	2110	2114

Structures are stored in Contiguous memory location

Date: / /

Q. Write a program to store the data of 3 students.

- Array benefit
~~For for making~~ for storing same types of data, we don't have to make different variable.

- Benefits of structure
1) Saves us from creating too many variables.
2) Good data management in an organization.

- Array of structures

```
struct student ECE[100];
struct student COE[100];
struct student IT[100];
```

Access

```
IT[0].roll = 200;
IT[0].cgpa = 7.6;
```

- Initializing structures

```
struct student s1 = {"bibek", 2003, 7.5};
struct student s2 = {"sanket", 2008, 8.3};
struct student s3 = {0};
```

8:47:30

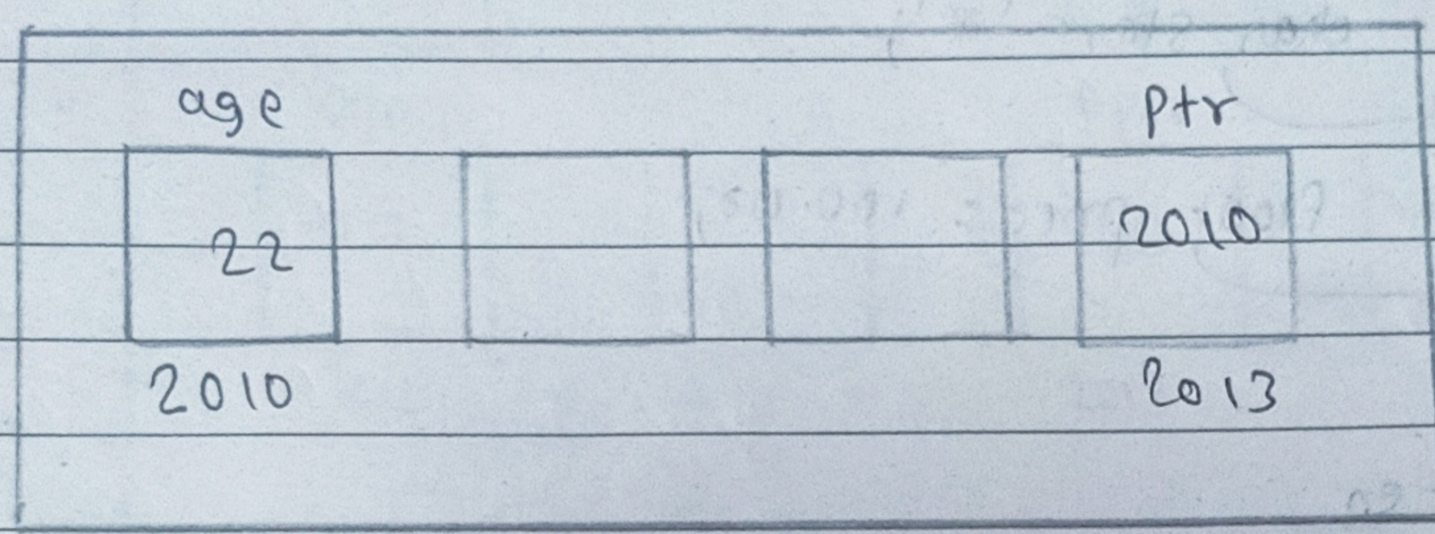
Page No.
Date: / /

Chapter - 6

Pointers

↓

A variable that stores the memory address of another variable



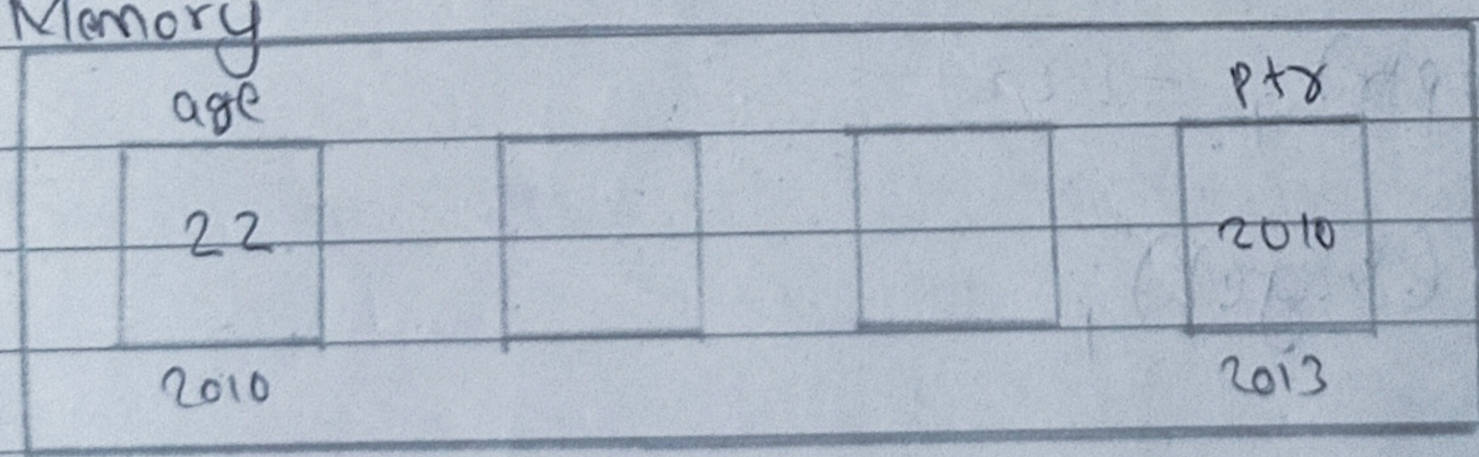
- Syntax

```
int age=22;
int *ptr = &age;
int _age = *ptr;
```

value at address operator

& address of operator

Memory



Date: / /

- Declaring Pointers

int *ptr; int age = 22;

char *ptr; char star = '*';

float *ptr; float price = 100.00;

- Format specifier

printf("%p", &age);

printf("%p", ptr);

printf("%p", &ptr);

printf("%d", age); → 22

printf("%d", *ptr); → 22

printf("%d", *(&age));

address type

%p → hexadecimal

pointer address

%u → unsigned int

- Pointer to Pointer

↓

A variable that stores the memory address of another pointer.

Memory

age		pptr	ptr
22		2013	2010
2010		2012	2013

- Pointer to Pointer

Syntax

```
int **pptr;

char **pptr;

float **pptr;
```

Print the value of 'i' from its pointer to pointer.

- Pointers in Function call

Call by Value	Call by Reference
we pass value of variable as argument	we pass address of variable as argument

- Q. Swap 2 numbers, a & b.

interchange

a = 3, b = 5

↓

a = 5, b = 3

a	b	t
3	5	

t = a
a = b
b = t

t = b
b = a
a = t

Q. 40 Write a function to calculate the sum, product & average of 2 numbers. Print that average in the main function.

~~We~~ When we want to ~~print~~ get many values we use pointer.

Page No.
Date: / /

Structures (Pointers to structures)

```
struct student s1;

struct student *ptr;

ptr = &s1;
```

- Arrow Operator

(*ptr).code ⟷ ptr->code

- Passing structure to function

```
// function Prototype
~~void printInfo~~ void printInfo(struct student s1);
```

- typedef keyword

↓

used to create alias for data types

```
typedef struct ComputerEngineeringStudents
    ~~int~~ int roll;
    float cgpa;
    char name[100];
} coe;

coe student1;
```

Q. Enter address (house no, block, city, state) of 5 people.

↓ int ↓ int ↓ char[] ↓ char[]

Page No.
Date: / /

File IO

File - containes in a storage device to store data

- RAM is volatile
- Contents are lost when program terminates
- Files are used to persist the data

- Operation on files

Create a file
Open a file
Close a file
Read from a file
Write in a file

- Types of files

Text Files	Binary Files
textual data	binary data
· txt, · c · java	· exe, · mp3, · jpg

- File Pointer

FILE
is a (hidden) structure that needs to be created for opening a file.

A FILE ptr that points to this structure & is used to access the file.

Question Solving C

Q. Write a program to change a given string to uppercase without using library function.

Q. Dissimilarities between structure and union.
→

	Structure	Union
Keyword	The keyword struct is used to define a structure.	The key word union is used to define union.
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Page No.
Date : / /

Q.1.-)
```
#include <stdio.h>
#include <string.h>

int main() {
    char str1[100];

    printf("Enter the string you want to convert to uppercase:\n");
    fgets(str1, sizeof(str1), stdin);

    for (int i=0, str1[i] != '\0', i++) {
        if (str1[i] >= 'a' && str1[i] <= 'z') {
            str1[i] = str[i] - 32;
        }
    }
    printf("The given string in uppercase = %s", str1);
    return 0;
}
```

Q. Write a program to display the Floyd's triangle.

```
1
2 3
4 5 6
7 8 9 10
```

Date: / /

```
-> #include <stdio.h>

int main() {
    int rows, cols, count;
    rows = 4;
    count = 1;

    for (rows = 1; rows <= 4; rows++) {
        for (cols = 1; cols <= rows; cols++) {
            printf("%d ", count++);
        }
        printf("\n");
    }
    return 0;
}
```

Q. C program to check if a number is a palindrome or not

```
#include <stdio.h>

int main() {
    int n, c, s=0, r;

    printf("Enter a number:\n");
    scanf("%d", &n);

    c=n;
    while (n>0) {
        r= n%10;
```

Page No.
Date: / /

```
    s = (s*10) + r;
    n = n/10;
  }
  if(c==s) {
    printf("palindrome\n");
  } else {
    printf("Not a palindrome");
  }
    return 0;
  }
```

- Initializing arrays

```
int marks[5] = {1,2,3,4,5}
char name[ ] = {'B','I','B','E','K'}
```

- Initialization of 2D array

```
int a[3][4] = {{0,1,2,3},{4,5,6,7},{8,9,10,11}};
```

3. Write a program to store and display basic information (Roll, name, address, email and phone) of n students of using array of structure.

Soln

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

~~printf("Enter the~~
struct student {
    char name[100];
    int roll;
    char address[100];
    char ~~ad~~email[100];
    int phone;
};

int main() {
    struct student s1;
    ~~str~~strcpy (s1.name, "Bibek shah");
    s1.roll = 03;
    strcpy (s1.address, "Dharan");
    strcpy (s1.email, "~~shah~~bibekshah78272@gmail.com");
    ~~pt~~ s1.phone = 9819110432;
    printf("name is %s", s1.name);
    printf("roll no is %d", s1.roll);
    printf("address is %s", s1.address);
    printf("phone no is %d", s1.phone);

    return 0;
}
```