unauthorized access.

Command Interpretation:

It tells care of interpreting user commands and directing the system resources to handle the requests

---

unit - III

(2) Overview of C:-

1) C is developed by Dennis Ritchie

2) C is a structured programming language

3) C is supports functions that enables easy maintainability of code, by breaking large file into smaller modules

4) comments in C provides easy readability

5) C is a powerful language.

2) Constant:-

1) It is a named value.

Ex:-

```
# define SIZE 100
# define PI 3.14
```

2) it cannot be reassigned a value.

variables / identifiers:-

1) it represent names of memory locations during program execution

2) Data values processed in the program are stored in memory locations and referenced through variables.

Rules of Naming variables (identifier)

1) A variable name in a sequence of characters each of which is choosen from the following:-

letters a, b, ... z ; A, B ..... Z

digits 0, 1, ... 9

underscore .

2) A variable name must start with a letter are an underscore.

3) it cannot be a keyword (reserved word)

Eg:- float, if, else, case, for etc.

4) Special characters (+, -, ?, ., ; ' ; " etc) are not allowed.

5) white spaces are not allowed.

3) Data type:-

1) it is a set of values & set of operations on those values.

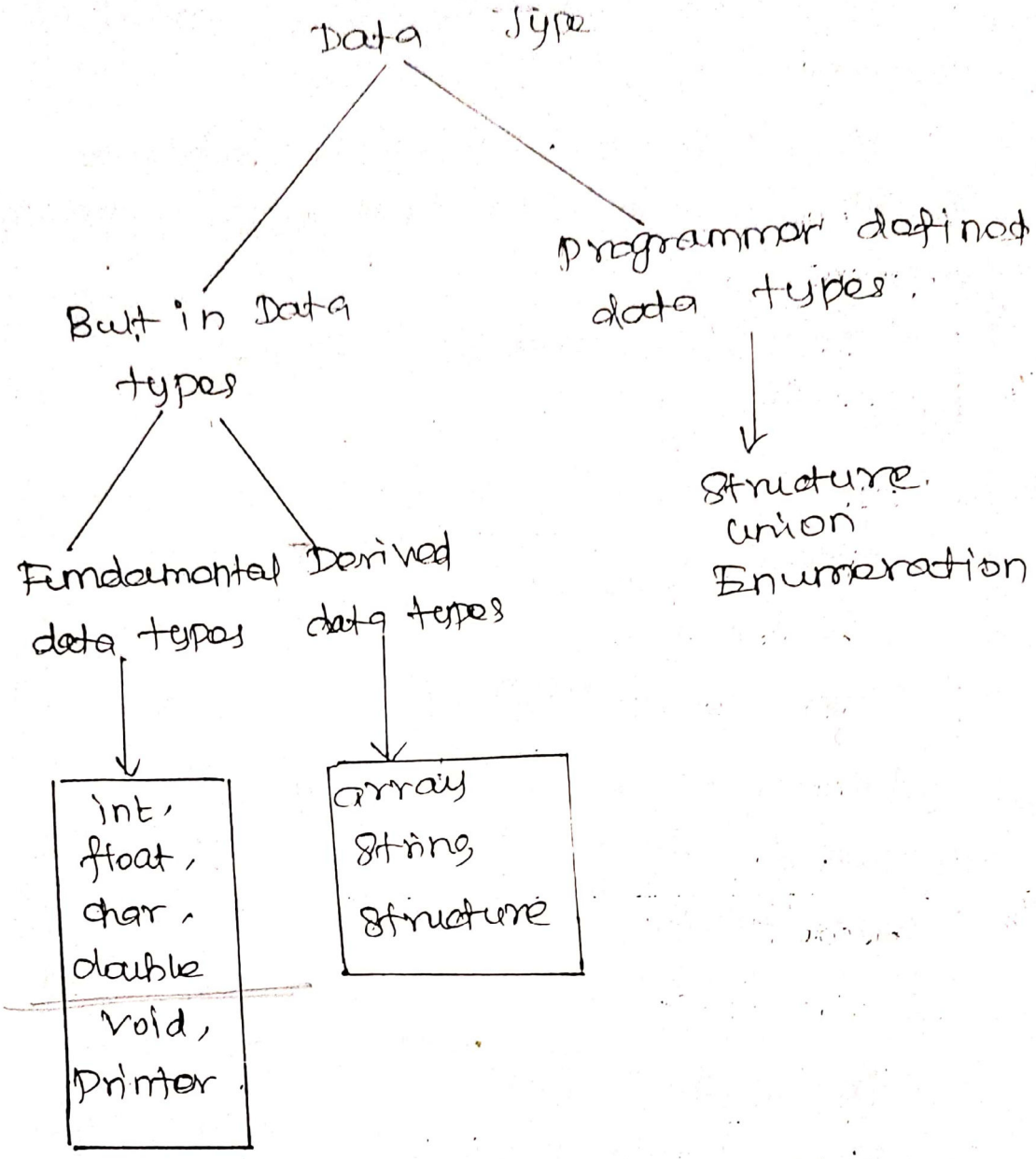
2) it is used to

(i) identify the type of a variable

(ii) identify the type of the return value of a function.

(iii) identify the type of a parameter expected by a function.

Type of Data types:-



Fundamental Data type:-

1) void - To denote the type with no values.

2) int - To denote an integer type.

3) char - to denote a character type

4) float, double - to denote a floating point type.

Derived Data type:-

Array:- a finite sequence of

String :-

an array of character variables.

5m Declaring variables :-

1) Any variable must be declared before it can be used in the program

Syntax :-

data - type variable name ;

Ex :-

```
int a, b, c
float x;
double y;
char name [10];
```

4) Structure of c program :-

5m

```
Headers files
main function
{
  Declaration part;
  Executable part;
}
```

Header files :-

1) The files that are specified in the include section called as header files.

2) We can call functions in our predefined program by supplying parameters.

3) main function is compulsory for any C program.

Eg:-

```
#include <stdio.h>
void main() include <Stdio.h>
{
printf ("welcome");
}
```

This program prints welcome on the screen when we execute it.

Running a C program:-

1) Type a program
2) Save it
3) Compile the program - generate exe file.
4) Run the program.

Comments in C:-

1) Single line comment - using //
2) Multiline comment - /* ..... */

5) operators in C:-

1) Arithematic operators
2) Assignment operators
3) Equalities and Relational operators.
4) Logical operators.
5) Conditional operators.

+, -, *, /, %.

C operation	operator	Example
Addition	+	a+3.
Subtraction	-	a-b
Multiplication	*	a*b
Division	/	a/c
Modulus	%	a%, x.

Assignment operators:-

Assignment operator is used to store the value into the variable.

Eg:- a=10, b=8.5 c='x'

Equality and Relational operators:-

1) 1) Equality operators:-

operator	Example	Meaning
==	x==y	x is equal to y
!=	x!=y	x is not equal to y

2) Relational operators:-

operator	Example	meaning
<	x<y	'x' is less than y
>	x>y	x is greater than y.
<=	x<=y	x is less than or equal to y.
>=	x>=y	y is greater than or equal to x

Logical operators:-

1) It is useful for test multiple Conditions.

2) Three types of logical operators.
AND, OR, NOT
&&, !!, ! in c language.

3) && - AND:-

i) all the conditions must be true for the whole expression to be true

eg:- if (sub1 >=40 && sub2 >=40 && sub3 >=40) prints ("pass")

4) !! - OR:-

(i) Any one condition in true the whole expression in true.

eg:- if (sub1 < 40 !! sub2 < 40 !! sub3 < 40) prints ("fail");

5 ! - NOT:-

(i) Reverse the meaning of a Condition.

eg: if (! (points >90)).

means if points not biggerthan 90.

Conditional operator:-

1) The Conditional operator (?:) is used to simplify an if/else statement.

Condition ? Expression 1 : Expression 2 ;

Eg: a > b ? printf ("a is big") ; : printf ("big").

6) Expression in c :-

Expression is a Collection of operands and operators.

Eg :- a+b*c

Types of Expression :-

1) Constant Expression
2) Integral Expression
3) Float Expression
4) Relational Expression
5) Logical Expression
6) Bitwise Expression.

Constant Expressions :-

it Consist of only constant values.

eg :- 15

```
int 5 / 2.0
!n'
```

Integral Expressions :-

it Produce the integer results

eg: m

```
m*n - 5
m* !n'
```

Float Expression :-

it produce the floating - point results

Eg :-

x * . y / 10

x + y

10 . 75

Relational Expression :- values

it produce the floating - point results only true or false

Eg :-

x && y ||

Eg :-

x <= y

a + b == c + d

m + n >= 100

Logical Expression :-

it combine two or more relational expressions and produces true / false results.

Eg : a < b && x == 10 .

Bitwise expressions :-

it is used to manipulate data at bit level, They are basically used for testing or shifting bits.

Eg :- x << 3 → shift 3 bit position to left.

y >> 1 → shift 1 bit position to right.

The file name **stdio.h** is an abbrevation for *standard input-output header* file. The instruction **#include <*stdio.h*>** tells the compiler 'to search for a file named **stdio.h** and place its contents at this point in the program'. The contents of the header file become part of the source code when it is compiled.

4.2 READING A CHARACTER

The simplest of all input/output operations is reading a character from the 'standard input' unit (usually the keyboard) and writing it to the 'standard output' unit (usually the screen). Reading a single character can be done by using the function **getchar**. (This can also be done with the help of the **scanf** function which is discussed in Section 4.4.) The **getchar** takes the following form:

***variable_name* = getchar( );**

variable_name is a valid C name that has been declared as **char** type. When this statement is encountered, the computer waits until a key is pressed and then assigns this character as a value to **getchar** function. Since **getchar** is used on the right-hand side of an assignment statement, the character value of **getchar** is in turn assigned to the variable name on the left. For example.

```
char name;
name = getchar();
```

Will assign the character 'H' to the variable **name** when we press the key H on the keyboard. Since **getchar** is a function, it requires a set of parentheses as shown.

Example 4.1 The program in Fig. 4.1 shows the use of **getchar** function in an interactive environment.

The program displays a question of YES/NO type to the user and reads the user's response in a single character (Y or N). If the response is Y or y, it outputs the message

My name is BUSY BEE

otherwise, outputs

You are good for nothing

NOTE: There is one line space between the input text and output message.

Program

```
#include <stdio.h>
main()
{
   char answer;
   printf("Would you like to know my name?\n");

   printf("Type Y for YES and N for NO: ");
   answer = getchar(); /* .... Reading a character...*/
```

```
if(answer == 'Y' || answer == 'y')
   printf("\n\nMy name is BUSY BEE\n");
else
   printf("\n\nYou are good for nothing\n");

}
```

Output

```
Would you like to know my name?
Type Y for YES and N for NO: Y

My name is BUSY BEE

Would you like to know my name?
Type Y for YES and N for NO: n

You are good for nothing
```

Fig. 4.1 *Use of* **getchar** *function to read a* **character** *from keyboard*

The **getchar** function may be called successively to read the characters contain line of text. For example, the following program segment reads characters from ke one after another until the 'Return' key is pressed.

```
--------

--------

char character;

character = ' ';
while(character != '\n')
{
    character = getchar();

}

--------

--------
```

WARNING

The **getchar()** function accepts any character keyed in. This includes RETURN and TAB. This means when we enter single character input, the newline character is waiting in the input queue after **getchar()** returns. This could create problems when we use **getchar()** in a loop interactively. A dummy **getchar()** may be used to 'eat' the unwanted newline character. We can also use the **fflush** function to flush out the unwanted characters.

NOTE: We shall be using decision statements like **if**, **if...else** and **while** extensively in this c ter. They are discussed in detail in Chapters 5 and 6.

Table 4.1 Character Test Functions

Function	*Test*
isalnum(c)	Is c an alphanumeric character?
isalpha(c)	Is c an alphabetic character?
isdigit(c)	Is c a digit?
islower(c)	Is c lower case letter?
isprint(c)	Is c a printable character?
ispunct(c)	Is c a punctuation mark?
isspace(c)	Is c a white space character?
isupper(c)	Is c an upper case letter?

4.3 WRITING A CHARACTER

Like **getchar**, there is an analogous function **putchar** for writing characters one at a to the terminal. It takes the form as shown below:

putchar ***(variable_name);***

where ***variable_name*** is a type **char** variable containing a character. This statemen plays the character contained in the ***variable_name*** at the terminal. For example, the ments

```
answer = 'Y';
putchar (answer);
```

will display the character Y on the screen. The statement

```
putchar ('\n');
```

would cause the cursor on the screen to move to the beginning of the next line.

Example 4.3 A program that reads a character from keyboard and then prin reverse case is given in Fig. 4.3. That is, if the input is upper cas output will be lower case and vice versa.

The program uses three new functions: **islower, toupper**, and **tolower**. The fu **islower** is a conditional function and takes the value TRUE if the argument is a low alphabet; otherwise takes the value FALSE. The function **toupper** converts the low argument into an uppercase alphabet while the function **tolower** does the reverse.

Program

```
#include <stdio.h>
#include <ctype.h>
main()
{
   char alphabet;
   printf("Enter an alphabet");
   putchar('\n'); /* move to next line */
   alphabet = getchar();
   if (islower(alphabet))
```

Formatted input :

Formatted input refers to an input data that has been arranged in a particular format.

e.g : 15.75 123 John

The lines contain three piece of data, arranged in particular form.

For example :

i) The first part of the data should be read into a variable "float".

ii) The second one "int".

iii) The third part into "char".

This is possible in C using the scanf function. [scanf means scan formatted]

The general form of scanf is,

```
scanf ("control string", arg1, arg2,
                        ..... argn);
```

Control string :

* The control string specifies the field format in which the data is to be entered and the argument arg1, arg2 ...., argn

specify the address of locations where the data is stored.

* "Control string and arguments are seperated by commas"

- Control string contains field specifications, which direct the interpretation of input data;

* Field specifications, consisting of the conversion character %, a data type character, and an optional number, specifying the field width.

* Blank, tabs, or newlines.

* Blanks, tabs and newlines are ignored.

Inputting integer number:

The field specification for reading an integer number is

% w sd

i) The percentage sign indicates %

Conversion specification

ii) w is a integr number that specified the field width.

iii) d known as data type character.

example: Scanf ("%2d %5d", &num1, &num2);

Data line:

50 31426

i) The value 50 will be assigned to num1.

ii) The value 31426 is num2.

31426 50

i) The variable num1 will be assigned to 31 (%.2d)

ii) The num2 will assigned to 426.

iii) The value 50 that is unread will be next scanf call.

The statement is,

Scanf ("%d %d", &num1, &num2);

The data:

31426 50

* 31426 will be assigned to num1

* 50 will be assigned to num2.

* Input data items must be seperated by spaces, tabs or newlines.
* Punctuation marks do not count as seperators.
* The fractional part may be stripped away!
* Also scanf may skip reading further input.

An input field may be skipped by specifying * in the field width.

e.g: scanf ("%d %*d %d", &a, &b)

Data: 123 456 789

i) 123 to a

ii) 456 will be skipped (because of *)

iii) 789 to b.

Inputting Real numbers:

Integer Number, the field width of real numbers is not to be specified and therfore scanf reads real numbers using the simple speaification %f for both the notations, namely, decimal point notation and exponential notation

The statement,

scanf ("%f %f %f", &x, &y, &z);

Input data line,

415.89 13.21E-1 618

* 415.89 will assign to x
* 1.321 will assign to y
* 618.0 will assign to z

* If the number to be read is double type
* then the specification should be %.lf instead of simple %f.
* A number may be skipped using %*f specification.

Inputting character strings:

* We have already seen how a single character can be read from the terminal using the getchar function.
* The same can be achieved using the scanf function.
* In scanf function can input strings containing more than one character.

For example of reading character strings:

%.ws or %.wc

* %c may be used to read a single character when the argument is a pointer to a char variable.

Some Version of scanf specification strings:

%[characters]

%[^characters]

* [%characters] are permissible in the input strings.

* [^%characters] are not permitted in the input strings.

Reading Mixed data type:

It is possible to use one scanf statement to input data line containing mixed mode data. The input data items match the control specifications in order and type.

Statement:

```
scanf("%d %c %f %s", &count, &code, &ratio, name);
```

data line:

```
15 P 1.575 coffee
```

* Some system accept integers in the place of real numbers and vice versa, the input data is converted to specified control string.

in the format string is necessary to skip the white space before p.

19

Formatted Output :

* It is used to print "f".

General form :

Printf ("control string", arg1, arg2 ..... argn);

Control string consists of three items :

1. characters will be printed on the screen.
2. Define o/p formed
3. Escape sequence characters such as \n, \t and \b.

Output of integer number :

%wd

* The % indicates conservation specification.
* w is a integer number that specified field width.

Output Of Real Numbers

The o/p of a real number can be displayed in decimal notation using the following specification

%w.pf

The integer w indicates the minimum number of positions that are to be used for the display of the value and integer p indicates the number of digits to be displayed after the decimal point. The value when displayed, is rounded to p decimal places and printed right justified in the field of w columns.

A real number can be displayed in the following exponential notation by using the specification %w.pe.

The display takes the form [-]m.nnnnne[±]xx

printing of a single character

A single character can be displayed in a desired position using the format:

%wc

The character will be displayed right justified in the field of w columns. We can make the display left justified by placing a minus sign before the integer w.

Printing of strings

The format specification for outputting strings is similar to that of real numbers. It is of the form

%w.ps

where w specifies the field width for display and p instructs only the first p characters of the string are to be displayed.

Mixed Data Output

It is permitted to mix data types in one printf statement. For example

printf ("%d %f %s %c", a, b, c, d);

Here printf uses its control string to decide how many variables to be printed and what their types are.