DATE / /
PAGE / /

040330 - PROGRAMMING IN C

UNIT - I

BASICS of C

SYLLABUS

1.1. Introduction to C :-

History of C - Structure of C Program - Steps for execution of C Program - Functions performed by Compiler, Linker - Algorithm & Flowchart - Low level and High level Programming language - C character set - Tokens - Constants - key words - Variables - Data types - Declaration of variables - Assigning values to variables.

1.2. I/O Statements :-

Formatted input, Formatted output, Unformatted I/O Statements.

SRIPC ECE

NOTES OF LESSON

UNIT - I

BASICS of C

1.1. Introduction to C :-

* History of C :-

→ Efficient and portable General Purpose Programming language.

→ Developed in the year of 1972 at Bell Telephone Laboratories (AT&T) by Dennis Ritchie.

→ C from second letter of earlier BCPL

→ BCPL - Basic Combined Programming Language developed by Ken Thompson at Bell Laboratories.

→ Until 1978 C - within Bell

→ Afterwards the language promoted.

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Features of C :-

→ Flexible high level structured programming language

→ Includes features of low level language like assembly language.

→ Portable. program written for one type of computer can be used in any other type.

→ Faster and Efficient

→ Has ability to extent itself

→ Has number of built-in functions

* Structure of a C program :-

→ The general structure of C program is,

SRIPC ECE

Include section or Header section

Global declaration section

main()

{

Local declaration section

Statement-1

Statement-n

}

User defined function section

REVOLUTION THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ..../..../....
PAGE: ..../..../....

* Include section or Header section:-
  - → The header files are included.
  - → These files depends on the library functions
  - → Must begin with # symbol.

  Example: #include <stdio.h>
  #include<math.h> etc,

* Global declaration section:-
  - → Declaration of variables which are common to both main block and function section.
  - → optional one.

* main() :-
  - → Must present in all programs.
  - → Execution starts from this.

SRIPC ECE

* Local declaration section:-
  - → Declaration of variables which are local to the main block.

* Statement:-
  - → valid C statements to solve the problem

* User defined function section:-
  - → optional
  - → Sub-programs by the user.

764 SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Example :- To find the Square of a number

```
#include< stdio.h>
#include< math.h>
```
} → Include or Header Section

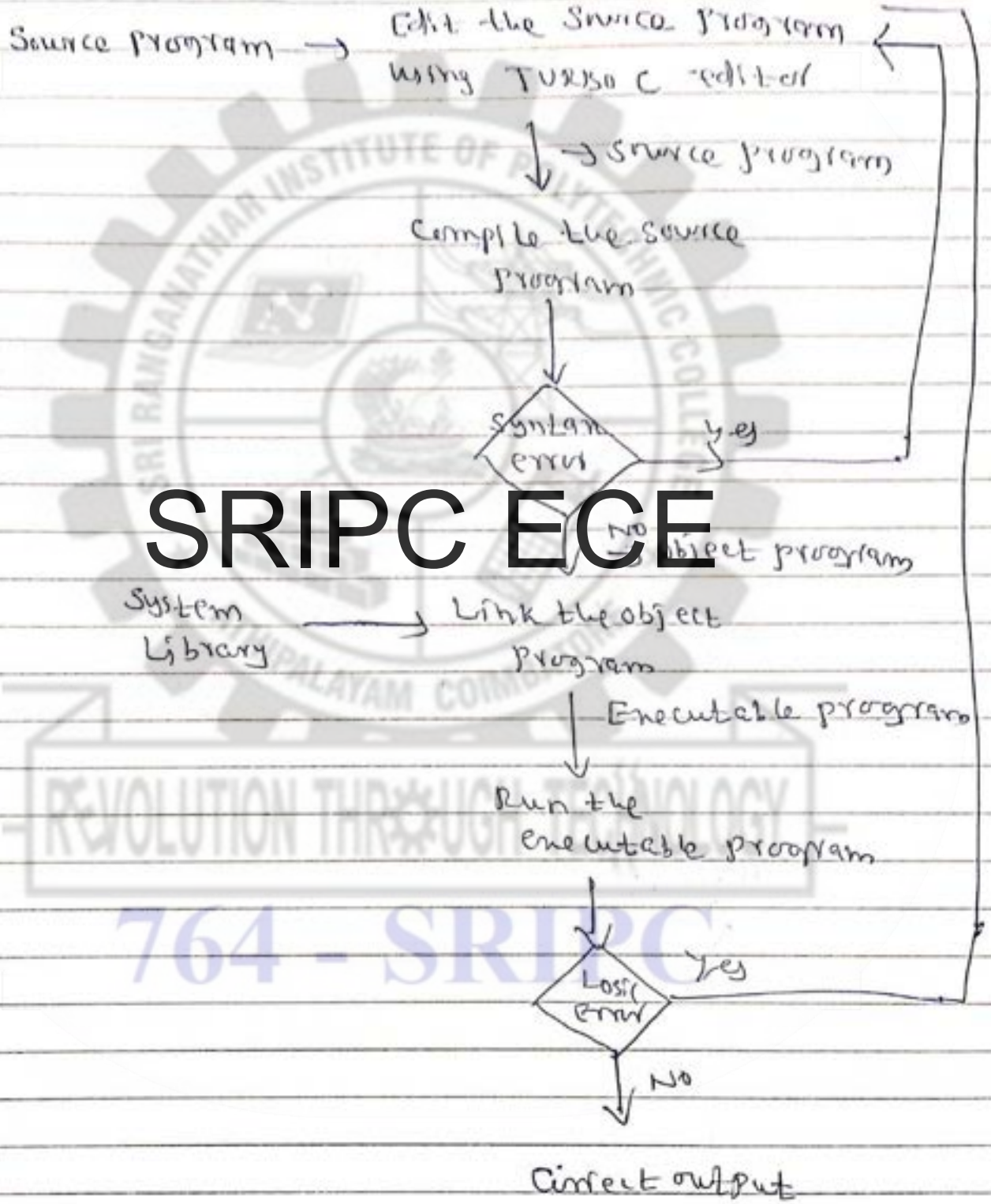
```
main()
{
float a, s, sq; → Local declaration Section

printf("enter the number");
scanf("%f", &a);
s = a * a;
sq = sqrt(a);
printf("Square = %f", s);
Printf("Square Root = %f", sq);
}
```
→ Statement section

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE / /
PAGE / /

* Diagrammatic representation of program Execution program



SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Execution of C program – using TURBO C Compiler

→ 3 Steps

1. Creating the program
2. Compiling the program
3. Linking and running the program

1. Creating the program

→ Double click TURBO C icon
→ The Editor window will open
→ Enter the program and Save the program
→ General form is,

filename.C

file name → name of the program
C → file extension
→ called source program

Example:

```
#include <stdio.h>
main()
{
 - - -
 - - -
 - -
}
```

→ Store as Sample.C

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

2. Compiling the program

→ Converts source program to machine language program.

→ Compile using compile option in editor.

→ If the program syntax error free, compiler translates it

→ The code called object program. Non executable

→ Compiler automatically names as

filename.obj

filename → Name of the program

obj → file extension

Example: Sample.obj

3. Linking and Running the object program:

→ Linking is also known as process of connecting other programs and functions used for our program.

→ Linking automatically performed.

→ Non executable object program converted into executable object program.

→ General form is,

filename.exe

filename → Name of the program

exe → file extension

→ Run the executable program using run option in TURBO C editor & get result.

→ If any logic error means correct the program & re-compile and link & run the program.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764-SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Algorithm :-
→ Step by step method to solve a problem using computer.

* properties :-
→ Finiteness :-
→ must terminate after a finite number of steps.
→ Each step can contain one or more operations.

→ Definiteness :-
→ Each step perfectly clear what should be done

→ Effectiveness :-
→ Each step solve by a person using pencil & paper in finite time

→ Input :-
→ Must accept zero or more i/p's.

→ Output :-
→ Must produce one or more o/p's.

Classification :-

* Backtracking algorithm
* Divide and Conquer algorithm.
* Greedy algorithm
* Branch and Bound algorithm
* Randomized algorithm.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

SRIPC ECE

Oval

Parallelogram → I/p or O/p data

Rectangle → processing

Yes / No → Decision making

Diamond

→ or ↓ or ↑ → flow direction

Arrow

→ Sub programs

Rectangle with double struck vertical edges

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Labelled Connector

→ To connect portions of flowchart drawn in diff. pages

Hexagon

→ for ... loop

Advantages:-

1. Communication :-
   → medium to communicate program
2. Effective analysis :-
   → helps to analyse the problem effectively
3. Proper documentation :-
   → helps to document the program
   → For future modification or error correction
4. Efficient coding :
   → Blue print during coding
5. Proper debugging :-
   → helps to debug the program

SRIPC ECE
764 - SRIPC
REVOLUTION THROUGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a,b,c,d,e,f;
  printf("enter a,b");
  scanf("%d%d",&a,&b);
  c=a+b;
  d=a-b;
  e=a*b;
  f=a/b;
  printf("The results are %d,%d,%d",c,d,e,f);
  getch();
}
```

SRIPC ECE

* Program :-

Sequence of instructions written to perform a well defined task with a Computer

* Program Development Cycle (PDC) :-

→ Sequence of steps followed to create a program

→ 7 steps

Problem Definition
↓
Problem Analysis
↓
Program Design
↓
Problem Coding
↓
Testing & Debugging
↓
Documentation
↓
Maintenance

764-SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a,b,c,d,e,f;
  printf("enter a,b");
  scanf("%d%d",&a,&b);
  c=a+b;
  d=a-b;
  e=a*b;
  f=a/b;
  printf("The results are %d,%d,%d",c,d,e,f);
  getch();
}
```

SRIPC ECE

* Program :-

Sequence of instructions written to perform a well defined task with a Computer

* Program Development Cycle (PDC) :-

→ Sequence of steps followed to create a program

→ 7 steps

Problem definition
↓
problem Analysis
↓
Program Design
↓
problem Coding
↓
Testing & Debugging
↓
Documentation
↓
Maintenance

764-SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Programming Languages :-

→ Language → Method for human communication
→ Uses words in a structured way
→ programming language → Method for communication between man and computer
→ Uses set of predefined words
→ 2 Types
  → Low level or Machine language
  → High level or human language

* Low level or Machine :-

→ Language known to computer
→ Uses stream of words made up of 0's and 1's
→ Machine dependent
→ Different types of computer use different machine language.

Ex: 0011 1101 1110 1111

→ Features :-
  → Translation not needed
  → Execution speed high
  → Difficult to program
  → Difficult to understand
  → Varies from Machine to Machine
  → Error correction very difficult.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ____/____/____
PAGE: ____/____/____

* High level or Human :-
  → Language known to programmer
  → Uses words as in English language.
  → Easy to understand

Example:
BASIC, FORTRAN, PASCAL, COBOL, C, C++, JAVA, VBASIC etc.,

Features:
→ Easy to program
→ Machine independent
→ Error correction easy
→ Execution time high
→ Needs translation, Before execution high level should be converted to Low level.

SRIPC ECE

* Variables, Constants and data types :-

* C Character set :
  → C uses following characters in order to develop its language.

(i). Alphabets – A... Z, a... z
(ii). Digits – 0, 1, 2, ... 9
(iii). Special Characters – ! , # , % , ^ , & , + , = , ~ , [ ]
* ( ) _ - \ | ; : ' "
{ } ' . @ $ < > / ?
(iv). White Space – blank space, horizontal tab, Carriage Return, new line, form feed
(v). Special Conditions – \b, \n, \t, \f, \r, \v, \a

764 - SRIPC
REVOLUTION THROUGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Keywords or Reserved words :-
  → words belonging to C Language
  → Have standard predefined meaning
  → Used for its intended purpose
  → Users have to right to change its meaning.
  → Should be written in lower case.

Example :-

auto	enum	sizeof
break	float	static
case	for	struct

SRIPC ECE

* Identifiers :-
  → Names given to variables, functions, arrays and other user defined objects.
  → User defined names

* Rules :-
  → Formed with alphabets, digits and a special Character (_).
  → First character must be an alphabet
  → No special Characters other than underscore allowed.
  → Case sensitive.
    AA is different from aa

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Example :-

~~vacfid~~

Valid → AO, BASICPAY, basicpay, TOTAL_PAY

Invalid → AB. → (.) not allowed

→ 9A → First should be alphabet

→ auto → Reserved word

* Constant :-

→ A quantity whose value does not change during program execution

→ 3 types

i. Numeric

ii. Character

iii. String

SRIPC ECE

i. Numeric :-

→ Made up of digits and some Special Charact

→ 2 types

1. Integer or fixed point

2. Real or floating point

1. Integer :-

→ Made up of digits without decimal point

→ Can have values from −32,768 to +32,767.

Rules :-

→ Formed with digits 0 to 9

→ Can preceded by + or − sign

→ No special characters allowed.

→ 3 types

1. decimal

2. octal

3. Hexa decimal

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Decimal integer constant :-

→ Made up of digits 0 to 9.

→ First should not be zero.

Example :-

Valid → 10, +10, −5420, 5743

Invalid → 5,734 – comma not allowed

→ 0452 – First digit should not be zero.

* Octal integer constant :-

→ Made up of digits 0 to 7.

→ First should be zero.

Example :-

Valid → 0, 0156, 0700

Invalid → 5743 – First should be zero

→ 08257 → 8 not allowed

→ 015.04 – Decimal point not allowed.

* Hexadecimal integer constant :-

→ Made up of digits 0 to 9 and alphabets A to F

→ First should be 0X or 0x.

Example :-

Valid → 0X532, 0X1F, 0X1f

Invalid → 23A – Begin with 0X

→ 0X53.35 – Decimal point not allowed

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Real Constant :-

→ Any number written with one decimal point.

Rules :-

→ Formed with 0 to 9 and decimal point

→ Can preceded by + or - sign.

→ No special characters used other than decimal point.

2 Forms

i. Fractional form

ii. Exponent form

i. Fractional form :-

→ A no. with one decimal point.

Example :-

valid - -0.5 [illegible]

Invalid - 15 [illegible] decimal point

- 16.47.57 - 2 decimal points not allowed.

ii. Exponent form :-

→ Used to express very large ~~of~~ and very small real constants.

→ General form,

mantissa e or E exponent

↓ ↓

either real or integer value — Integer value

Rules :-

→ Mantissa and exponent can have positive or negative sign.

→ Exponent should not have decimal point

→ Exponent must have atleast one digit

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Example :-

0.0000025711 Can be expressed as .2571E-5

-0.00000000000 Can be expressed as -47E-11 or -4.7E-10.

Invalid :-

.840E.5 - Exponent should not have decimal point

So - Either a decimal point or an exponent must be present

* Character Constant :-

→ 2 types

1. Direct

2. Escape Sequence

SRIPC ECE

1. Direct :-

→ Contains a single character enclosed within single quotation marks.

→ Gives Integer value of the enclosed character

→ This value is known as ASCII value.

Example :-

Constant	Value
'A'	- 65
'0'	- 48
'%'	- 37

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

2. Escape sequence :-

→ More than one character within single quotation marks.

→ First character must be backslash (\).

→ Even though it has more than one character it represents only one.

Example :-

Meaning	Constant
backspace	'\b'
newline	'\n'
horizontal tab	'\t'
vertical tab	'\v'

SRIPC ECE

String Constant :-

→ Sequence of characters enclosed within double quotation marks.

→ Formed by digits, alphabets and Special Characters.

Example : "NAGERCOIL", "12573"

* Difference between character constant and String Constant

→ Character constant produces numeric value of the character enclosed in single quotes

→ String constant is a sequence of characters enclosed within double quotes.

'B' is not same as "B".

764 - SRIPC
REVOLUTION THROUGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Variables :-
→ A quantity whose value changes during the execution of the program.
→ C variable → Name given to the memory location to store data
→ particular memory location is given only one name.

* Rules :-
→ Formed with alphabets, digits and a special character underscore (_)
→ First character alphabet
→ No special characters other than (_) underscore.
→ Both uppercase and lowercase letters are used. But they are not treated as same
→ should not be a reserved word.

Valid : A0, BASIC_DAY, Volume
Invalid : AB. → . not allowed
9A → First must be alphabet
Go to → Reserved word.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../....
PAGE: ../../....

* Tokens :-

→ Smallest individual elements in C.
→ C program written using available tokens
→ More than one token can appear in a single line
→ Each token can be separated by white spaces.
→ 6 types of tokens.

i. keywords
ii. Identifiers
iii. Operators
iv. Constants
v. Strings
vi. Special symbols.

Example :-

```
main()
{
int a,b,c;
...
```

This program segment contains the following 11 tokens :

```
main ( ) { int a, b, c ;
```

* Data types and Storage :-

→ Supports different data types
→ Programmer can select.
→ Storage representation differ from machine to machine

* The classes of data types :-

1. Basic or fundamental
2. User defined
3. Derived

1. Basic :-

→ Already defined
→ Supported by C compiler

SRIPC ECE
764-SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Data type	Description	Storage requirement
1. Char	A character in the Character set	1 byte or 8 bits
2. int	An integer	2 bytes or 16 bits
3. float	a single precision floating point number	4 bytes or 32 bits
4. double	a double precision floating point number	8 bytes or 64 bits.

* Data type qualifiers :-
  - → Keywords [illegible] with the basic data type.
  - → Alters size or sign.
  - → Different qualifiers
    - 1. Signed, 2. Unsigned → sign qualifiers → affects sign
    - 3. Long, 4. Short → size qualifiers → affects size

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: / /
PAGE: / /

* Declaring Variables :-
  → All variables should be declared before it is used.
  → Done by declaration statement
  → The general form,
    data-type variable list ;
    ↓ ↓
  Valid data type — List of variables separated by comma.
  int, char etc.,

Example :-
  i. int A, B, C ;
    → Declares A, B and C as integer variables and allocates memory space.
  ii. Char name ;
  iii. float avg ;
  iv. Char name[4] ;
    ↓
    String variable

Uses :-
  → name to a memory location
  → Allocate memory space.

* Initializing Variables :-
  → Assigning initial values to variables during declaration.
  → The general form is,
    data-type variable = initial value ;

Example :-
  int all = 100 ;
  → all is a integer variable & assigns 100 initial value.

SRIPC ECE
764 - SRIPC
REVOLUTION THROUGH HIGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Assigning values to variables :-

→ Giving values to variables

→ = assignment operater used.

→ The general form,

Variable_name = value;

↓

Valid used defined name & should be declared.

Example :-

1. n=20
2. PI = 3.14

* Declaring variable as constant :-

→ A constant is a quantity whose value does not change during program execution.

→ General form,

const datatype Variable = value;

↓ ↓

Keyword valid data type (int, float)

Example :-

const float PI = 3.14;

The value 3.14 will not change during program execution

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

* Declaring Variables as Volatile :-

→ The general form,

volatile datatype variable;

↓ ↓

Keyword valid type (int, float)

→ This tells the value may change at any time by external source

Example :-

volatile int x;

* Input / output functions (Statements)

→ I/o functions used to transfer information b/w PC & I/o devices.

→ All the functions present in header file <stdio.h>

→ 2 types

1. Formatted I/o function
2. Unformatted I/o function

1. Formatted input function :-

→ scanf() function - to give data to variables

→ general form is,

scanf("control string", &variable1, &variable2, ... &variablen);

→ control string gives format of data

→ general form,

%w data-type

% - Conversion specification indicator

w - width of i/p data

data-type - type of data

SRIPC ECE
704 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

→ & symbol – address operator
  → Evaluates address of the variable
  → no & symbol for string data.

Rules :-
→ Variable names separated by comma
→ No. in specification & control string equal
→ specification continuous or separated by blank space
→ Variable name preceded by & symbol

Conversion Character table :-

Character	Meaning
c	single character
d	decimal integer
e or f or g	floating point value
h	short integer
i	decimal / hexa decimal / octal integer
o	octal integer
s	string

SRIPC ECE
REVOLUTION THROUGH HIGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Example:

i. int a;
scanf("%d", &a);
→ Reads integer & assigns to a

ii. float a;
scanf("%f", &a);
→ Reads float & assigns to a

iii. int a,b;
float c;
scanf("%d %d %f", &a, &b, &c);

iv. Char ch;
scanf("%c", &ch);

v. Char name[30];
scanf("%s", name);

vi. int a;
scanf("%4d", &a)
→ width of i/p data limits to 4 digits

* Formatted output function:-

→ printf() function used
→ General form,
printf("control string", list);
↓
Gives format of data

%w.p data-type

SRIPC ECE
764 - SRIPC
REVOLUTION IN HIGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

% - Conversion specification indicator
w - width of output data
p - no. of digits after decimal point
data-type - type of o/p data
→ list - list of variables, constants, array names.

1. Printing integer :-
→ General form,
%wd
w - width of data
d - Conversion character for integer

SRIPC ECE

Rules :-
→ If no width specified no printed as such
→ If o/p no. width is > then specified then it will be printed in full
→ No. printed is right justified. Left by - sign after % character
→ If o/p no. width is < then specified then the unused left blank
→ + or - sign the sign should be placed before the variable.

Ex :
x = 1999.

764 - SRIPC
REVOLUTION THROUGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ...../...../.....
PAGE: ...../...../.....

1. printf("%d", x); 1999
2. printf("%-3d", x); 1999
3. printf("%5d", x); 1999
4. printf("%-5d", x); 1999
5. printf("%07d", x); 0001999
6. printf("%-4d", -x); -1999

2. Printing real numbers:
→ General format,
%w.p f or e
w - width of output
p - digits after decimal point
f - Conversion character without exponent
e - " " with "

Rules:-
→ If no decimal place count is placed, the default is 6 places.
→ If no. has more than 5 decimal places, the result rounded to six places.
→ Integer part - right justified
Decimal part - left justified.
→ By placing - sign after % character the integer part left justified.

Ex:- x = 123.4678
1. printf("%8.4f", x); 123.4678
2. printf("%f", x); 123.467800
3. printf("%8.2f", x); 123.47
4. printf("%e", x); 1.234678e+02
5. printf("%-8.2f", x) 123.47

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Printing strings :-

→ The general format of control string,

%o W.P S

W – width of the string

P – no. of characters to be printed from the beginning

S – conversion character for string

Rules :-

→ width of string > width specified, the string will be printed in full.

→ The o/p is right justified.

→ – sign after % character left justified.

SRIPC ECE

Example : Nagercoil

printf("%s", place); Nagercoil

printf("%.5s", place); Nager

printf("%5s", place); Nagercoil

printf("%-5s", place); Nagercoil

printf("%10.5s", place); Nager

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ...../...../.....
PAGE: ...../...../.....

3. Unformatted Input function:-

i. getchar()
ii. gets()

(i). getchar()

→ Read a single character
→ Variable name = getchar();
↓
User defined character type name

Rules:-
i. Variable name declared previously
ii. Variable name character type
iii. Empty parentheses must
iv. Should include stdio.h header file.

Example:-
Char n;
n = getchar();
Character assigned to variable n

SRIPC ECE

(ii). gets();

→ Read a string of characters
→ gets(variablename);
↓
Valid C variable name

764 - SRIPC

Rules:-
→ Variable name declared previously
→ " array of character type
→ Should include stdio.h header file.

Example:-
Char name[10];
gets(name)

Protected with free version of Watermarkly. Full version doesn't put this mark.

3. Unformatted Input function:-
   i. getchar()
   ii. gets()

(i). getchar()
   → Read a single character
   → Variable name = getchar();
                ↓
        User defined character type name

Rules:-
   i. Variable name declared previously by
   ii. Variable name character type
   iii. empty parentheses must
   iv. Should include stdio.h header file.

Example:-
   Char n;
   n = getchar();
   Character [illegible] Variable n

SRIPC ECE

(ii). gets();
   → Read a string of characters
   → gets(variable name);
                ↓
        Valid C variable name

Rules:-
   → Variable name declared previously by
   → " array of character type
   → Should include stdio.h header file.

Example:-
   Char name[10];
   gets(name)

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Unformatted output functions :-

i. putchar()
ii. puts()

i. putchar();

→ Display a single character
→ putchar(character variable);

Rules :-

→ Character variable declared previously
→ Value should be assigned to variable before putchar() function
→ stdio.h should be included

Example :-

```
Char n;
n = getchar();
putchar(n);
```

ii. puts();

→ To display strings

puts(variable name);

↓

Valid C variable

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

Rules:-
→ String value should be assigned
→ stdio.h to be included.

Example:-
```
char name[4];
gets(name);
puts(name);
```

* Program:-
Sequence of instructions written to perform a well defined task with a Computer.

* Program development cycle:-
⇒ Set of steps to develop a pgm.
→ Phases

SRIPC ECE

→ Problem Definition (Define prob)
↓
Requirement Analysis (I/P, o/p, rules)
↓
Program design (Flowcharts, record layouts, overall plan)
↓
Problem coding (design to code)
↓
Testing & debugging (error finding & correcting)
↓
Documentation (recording)
↓
maintenance (to rectify error
to ↑ efficiency
Policy changes)

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, s;
scanf("%d", &a);
s = sq(a);
printf("%d", s);
getch();
}
void sq(int)
{
int b, sq;
sq = b * b;
return;
}
```

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

2. Unary :-

→ needs only one operand

↳ Uo operand

↓ ↓

Unary operater — Valid constant or variable

Table :-

Operater	operation	Example
—	Unary minus	—10
++	Increment	++i
--	Decrement	--j

2. Relational operators :-

→ To find relationship b/w 2 operands

→ Uo operand / 2o operands

↓ ↓

Relational operater — valid constant or variable or arithmetic expressions

Table

operater	operation	Example
>	greater than	A > B
<	Less than	A < B
>=	greater than or equal to	A >= B
<=	Less than or equal to	A <= B
==	equal to	A == B
!=	not-equal to	A != B

SRIPC ECE
764-SRIPC
REVOLUTION THROUGH TECHNOLOGY
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ..../..../....
PAGE: ..../..../....

3. Logical operators:-

→ To find relation between relational expressions

→ General form,

operand 1 Lo operand 2

↓ ↓

Logical operator Valid relational expression

Operator	Meaning
&&	AND
||	OR
!	NOT

x	y	x&&y	x||y
0(F)	0(F)	F	F
0(F)	1(T)	F	T
1(T)	0(F)	F	T
1(T)	1(T)	T	T

x, y – Relational expressions

T – True value 1

F – False value 0

4. Increment and decrement operators :-

→ For control loop

1. Increment operator :-

++

Adds 1 with value of variable

Variable++ or ++variable

Uses & increment	Increment & use
1. a=a++ , a=a+1	3. a=b++ , a=b b=b+1
2. a=++a , a=a+1	

SRIPC ECE
KUMARAGURU INSTITUTE OF POLYTECHNIC COLLEGE, PIPALIYAM : COIMBATORE
KNOWLEDGE THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

2. Decrement operater :-

-- subtracts 1 from value of variable.

General form,

variable -- or -- variable

Uses & decrement decrement & uses

Example:

--a , a=a-1, a-- ≠ a=a-1

5. Short hand assignment operater :-

→ simplify statement

→ variable operater = expression

Valid use of short hand operater

name

operater	Example
+ =	x=x+10, x+=10
- =	x=x-10, x-=10
* =	x=x*10, x*=10
/ =	x=x/10, x/=10
%=	x=x%10, x%=10

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

6. Conditional operator :-

→ ? & : to build conditional expression

→ Ternary operator

expression1 ? expression2 : expression3 ;

1 evaluated
If true 2 evaluated
If false 3 evaluated.

Ex : big = a>b ? a : b ;

a>b, tested, if true big = a, else big = b.

7. Bitwise operater :-

→ bit by bit operation

Operator Meaning

&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Bitwise right shift
<<	Bitwise left shift
~	Bitwise Complement

* Bitwise AND, OR and EX OR

Bit from pattern1	Bit from pattern2	AND	OR	Ex-OR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Bitwise left shift (<<)

→ Shift bit left

→ Contains

1. bit pattern

2. No. of shift

→ General form,

bit pattern << number of shift

x = 0 0 1 1 0 0 1 1

x<<2 = 00110011 << 2

= 0 0 1 1 0 0 1 1 0 0

shifted out

filled

SRIPC ECE

x<<2 = 11001100

* Bitwise right shift (>>)

→ Shift bit right

→ Contains

1. bit pattern

2. No. of shift

→ General form,

bit pattern >> number of shift

x = 0 0 1 1 0 0 1 1

x>>2 = 00110011 >> 2

= 0 0 0 0 1 1 0 0 1 1

bits filled

bits shifted out

x>>2 = 00001100

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: __/__/____
PAGE: __/__/____

* Bitwise Complement (~)
  → Changes all zeros to one and all ones to zero.
  ~ bit pattern
  Example:
  Let n = 1100
  ~n = 0011

8. Special operater :-
  1. Comma
  2. Size of
  3. Dot
  4. Address
  5. Indirection
  6. Arrow

SRIPC ECE

1. Comma :-
  → Used to link related expressions together
  → variable = (expr1, expr2, ... exprn)
  variable: defined variable. expr: valid expression
  Ex: sum = (a=5, b=20, c=10, a+b+c);

2. Size of operater :-
  → Used to find number of bytes occupied by operand
  → variable = sizeof (operand);
  Ex: int sum;
  int m = sizeof(sum)
  O/p is 2 bytes

REVOLUTION THROUGH TECHNOLOGY
764-SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

3. Dot operater :-
→ Used to give data to the structure variables individual members.
→ General form,
Structure variable. membername
Ex: Student1. age

4. Address operater(&);
→ Address of variable got by Address operater
→ &variable.
Ex;
int n;
printf("%d",&n)
Address of variable n printed.

SRIPC ECE

5. Indirection operater (*)
→ Used to access value stored in variable using pointer variable.
→ * pointer variable.
↓ ↓
Indirection operater — declared pointer variable.

REVOLUTION THROUGH TECHNOLOGY
764 SRIPC

6. Arrow operater (→)
→ Used to access Structure members
→ Structure variable → member name

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

* Assignment Statement :-
→ Used to assign values to Variables
→ Variable = Constant or Variable or Expression
En:
x = 10
y = x + 10 * y

* C - Expressions :-
→ Linear Combination of Constants, Variables and operators.
→ 3 types
1. Arithmetic
2. Relational
3. Logical

1. Arithmetic :
→ Formed by connecting Constants or variables by arithmetic operators.

Constant or Variable	Arithmetic operator	Constant or Variable

Example : x+2, (x+y)/2

2. Relational expression :
→ Formed by connecting Constants or variables or arithmetic expressions by relational operators.
General form,

Constant or Variable or expression	Relational operator	Constant or Variable or expression

En:
A == 20 A * A - B > C * C - D

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

3. Logical expression:-
→ Formed by connecting relational expressions by logical operators.
→ General form,

Relational expression | Logical operator | Relational expression

Example:
1. A>20 && B>25
2. A>20 || B>25

Arithmetic expression:-
→ 3 types
1. Integer
2. Real
3. [illegible]

SRIPC ECE

1. Integer:-
→ If datatype of i/p & o/p variable is integer means.

→ int var1 = int var2 & int var3 ....
Ex:
int a;
int b=10, c=20;
a=b+c.
a is 30.

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ..../..../....
PAGE: ..../..../....

2. Real expression:-
→ If data type of i/p & o/p variable is float means.
→ float var1 = float var2 & float var3
...

Ex:
float a;
float b=15.5, c=19.5
a = b+c
a is 35.0

3. Mixed mode expression:-
→ If data type of i/p & o/p variable is different means
→ data type var1 = data type var2 & data type var3

Ex:
int a,b;
float c,d;
d = a+b-c;

* Type conversion in arithmetic expression:-
→ process of converting operands in an expression from one data type to another
→ 2 types
1. Automatic
2. Explicit

1. Automatic:-
→ lower data type operand automatically converted to higher data type.
Ex: float a, b=5.7
int c=10;
a = b+c
a is 10.0+5.7 = 15.7

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

2. Explicit :-

→ Converting one data type to another locally without affecting its data type in the declaration.

→ (data type) variable;

Ex: x = (int) 7.5;

7.5 is converted to 7.

* Operator precedence and Associativity :-

→ The priority given to an operator during evaluation.

→ The association of an operator with its operands called associativity.

* Evaluation of expression :-

The order was,

1. inside parenthesis
2. Unary operators
3. Multiplication, division, modulo operators
4. Addition & subtraction
5. relational & logical operators.
6. assignment operators

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Decision making, branching & looping statements :-

* Decision making and branching :-

→ Skip or to execute group of statements based on the result of some condition.

1. Simple if
2. if...else
3. else...if Ladder
4. nested if...else
5. switch

1. Simple if Statement :-

→ Used to execute or skip one statement or group of statements if a particular condition

→ General form,

if (test condition)

{

statement block;

}

next statement;

Test evaluated

True means statement block

False " " skipped.

Rules :-

→ Brackets around test condition must

→ Test condition must be relational or logical expression

→ Statement block contains one or more statements.

→ { , } need if more than one statement

SRIPC ECE

EVOLUTION THROUGH TECHNOLOGY

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

Flow diagram

Program :-

```
#include <stdio.h>
main()
{
int mark;
char grade;
scanf("%d %c", &mark, &grade);
if (grade == 'A')
{
mark = mark + 10;
}
printf("%d", mark);
}
```

Tests the grade
if A, adds 10 & print
else print mark as such.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: / /
PAGE / /

2. If ... else statement :-

→ Used to execute one group of statements if the test condition is true or other group if the test condition is false.

→ General form,

```
if (test condition)
    {
    statement block-1;
    }
else
    {
    statement block-2;
    }
next statement;
```

SRIPC ECE

→ Evaluates test condition

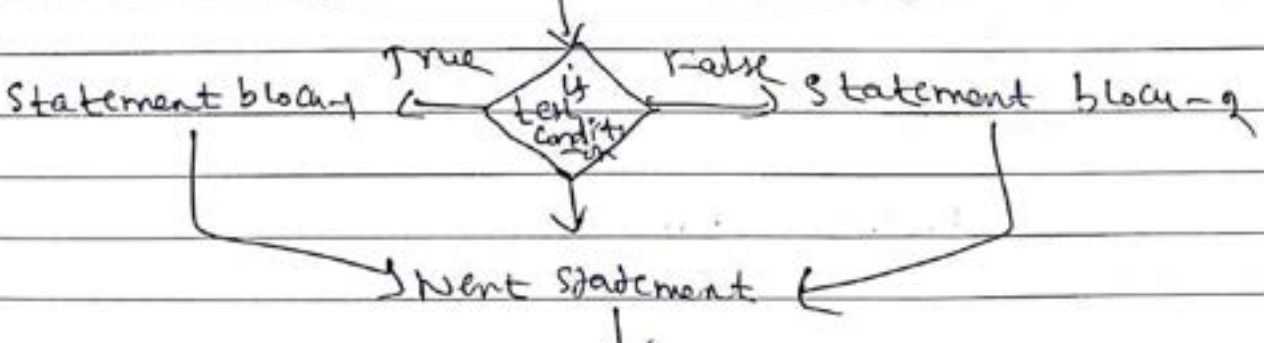
→ If true statement block-1 is executed & control transfer to next statement

→ If false statement block-2 is executed & control transfer to next statement

Rules :-

* Brackets around test condition must.
* Test condition relational or logical expression
* Statement block called body of if - else statement
* opening & closing { } needed.

Flow diagram.



Protected with free version of Watermarkly. Full version doesn't put this mark.

Example :-

```
#include <stdio.h>
main()
{
 int mark;
 scanf("%d", &mark);
 if (mark >35)
   printf("pass");
  else
   printf("fail");
}
```

Tests the mark of the student
if it is greater than 35 it prints pass
" less " " fail.

SRIPC ECE

3. else... if ladder statement :-

- multiway decision.
- Formed by using if...else statements in which each else contains another if...else.
- The general form is,

```
if ( test condition-1)
 {
  statement block-1;
 }
else if ( test Condition-2)
 {
  statement block-2;
 }
```

Protected with free version of Watermarkly. Full version doesn't put this mark.

```
else if ( test condition-n )
  {
   statement block-n;
  }
 else
 default statement;
}

next statement;
```

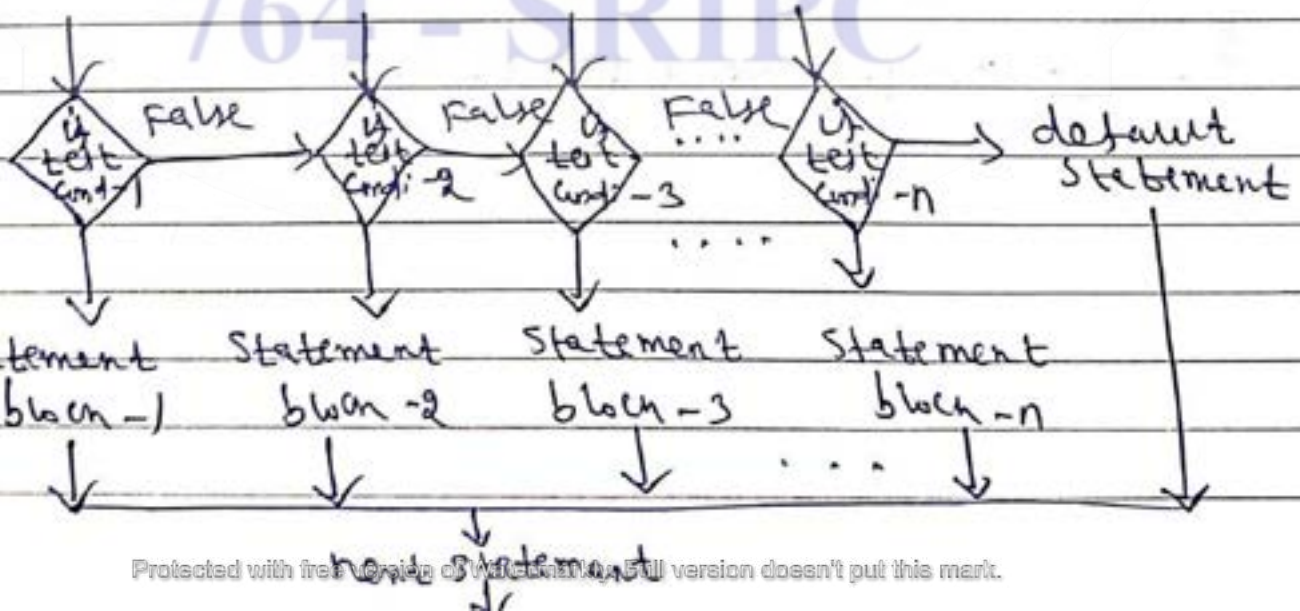
- Execution top to bottom
- if true
  true block executed then next statement
- All test condition false, final else containing default statement executed

SRIPC ECE

Rules

→ brackets around test condition must
→ test condition relational or logical expression
→ statement block – one or more statements
→ opening and closing brackets { } must if more than one statement.
→ default statement must.

Flow diagram :-



764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Example :- To display the day, depending upon the number entered.

```
#include <stdio.h>
main()
{
int day;
printf("enter number between 1 and 7");
scanf("%d", &day);
if (day==1)
  printf("Monday\n");
else if (day==2)
  printf("Tuesday\n");
else if (day==3)
  printf("Wednesday\n");
else if (day==4)
  printf("Thursday\n");
else if (day==5)
  printf("Friday\n");
else if (day==6)
  printf("Saturday\n");
else if (day==7)
  printf("Sunday\n");
else
  printf("enter correct number");
}
```

SRIPC ECE
EVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

4. Nested if...else Statement:-

→ Joining if...else statement either in the if block or in the else block or both.

→ The general form is,

```
if (test Condition-1)
  {
  if (test Condition-2)
    {
    Statement block-1;
    }
  else
  {
  Statement block-2;
  }
else
{
Statement block-3;
}
next statement;
```

SRIPC ECE

→ test condition-1 evaluated
→ if true test condition-2 evaluated
→ if false statement block-3 executed.
→ if test condition-2 true, statement block-1 executed
→ if " false, " " -2 "
~~→ exit~~

Rules:-

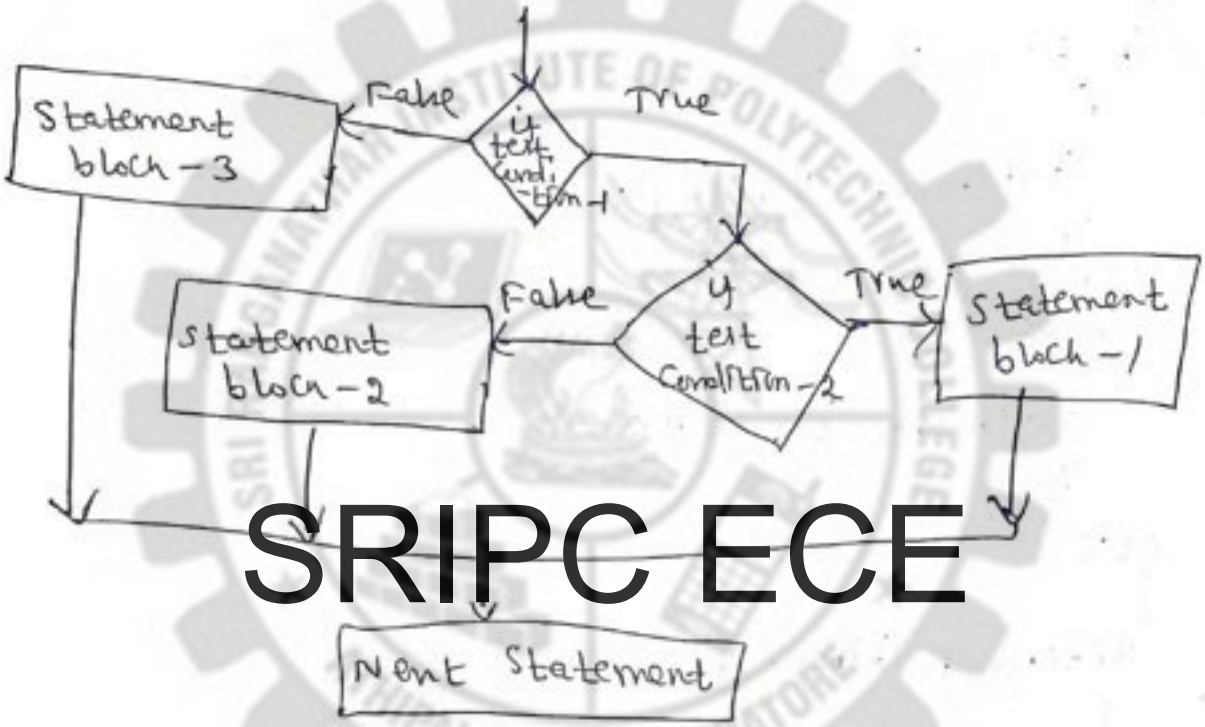
→ Brackets around test condition
→ Test condition, relational or logical expression
→ Statement block — one or more statements
→ Opening and closing brackets { } must if more than one statement

Protected with free version of Watermarkly. Full version doesn't put this mark.

→ NO statements are allowed other than Statements in statement block

→ else match with if

Flow diagram :-



SRIPC ECE

Example : Program to find the biggest of 3 numbers.

```
#include <stdio.h>
main()
{
int a,b,c;
scanf("%d %d %d", &a, &b, &c);
if(a>b)
    {
     if(a>c)
       printf("big=%d", a);
     else
       printf("big=%d", c);
    }
else
  {
```

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

```
if(b>c)
 printf("big = %d", b);
else
 printf("big = %d", c);
 }
}
```

→ Used to find out biggest of 3 numbers.
→ if...else statements included in the if block and else block

V. Switch statement:-
→ Extension of if...else statement
→ Permits any number of branches
→ General form,

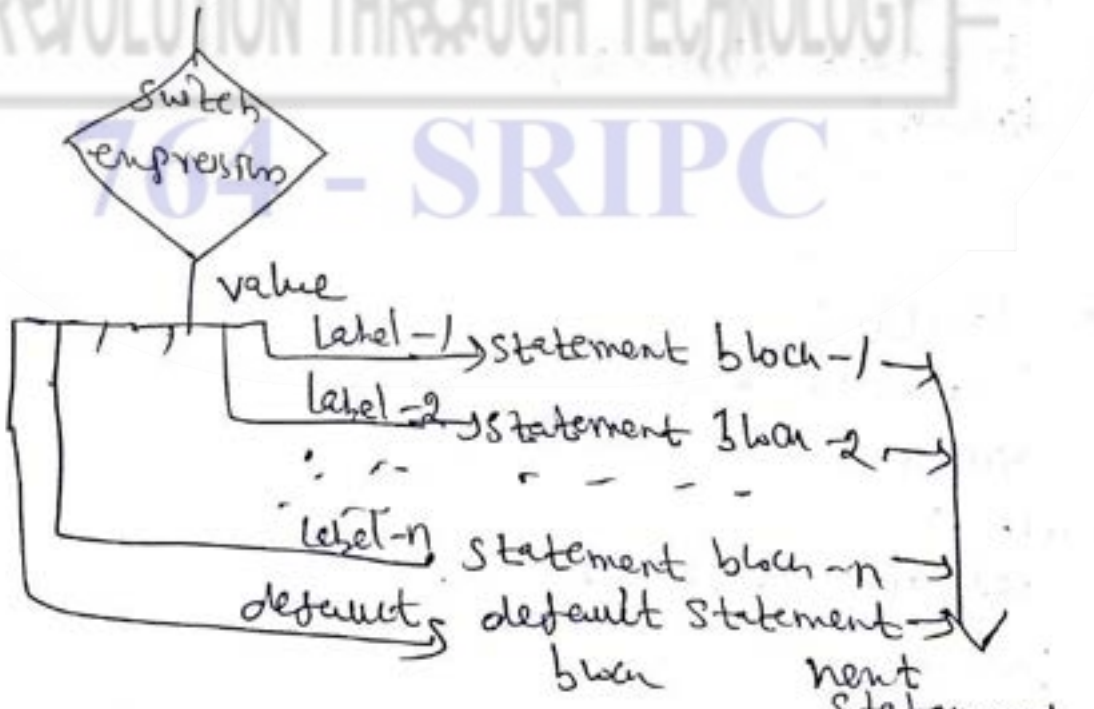
```
switch(expression)
{
 case label 1:
      statement block-1;
      break;
 case label 2:
      statement block-2;
      break;
 - - - - -
 - - - - -
 case label n;
      statement block-n;
      break;
 default:
      default statement;
      break;
}
next statement;
```

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Printed with free version of Watermarkly. Full version doesn't put this mark.

→ Evaluates the expression.
→ Value compared with case label 1, Label 2, ... Label n
→ If case label matches with value, the Statement block executed
→ Cntrl transferred to next statement.
→ If none of case matches, default Statement block executed.

Rules :-
→ Expression placed in parentheses
→ Value of expression — integer
→ Body of switch statement enclosed [illegible]
→ [illegible] terminate with a colon
→ Break must, else statements below matched case will be executed.

Flow diagram :-



SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Example:
To display day depend on number entered.

```
#include <stdio.h>
main()
{
int day;
scanf("%d", &day);
switch(day)
{
case 1:
printf("Monday\n");
break;
case 2:
printf("Tuesday\n");
break;
case 3:
printf("Wednesday\n");
break;
case 4:
printf("Thursday\n");
break;
case 5:
printf("Friday\n");
break;
case 6:
printf("saturday\n");
break;
default:
printf("Sunday\n");
break;
}
}
```

SRIPC ECE
SRIPC - 704

Protected with free version of Watermarkly. Full version doesn't put this mark.

Unconditional goto Statement :-

→ transfer program control unconditionally from one point to another.

→ The general form,

goto label;

Label is the name given to the point where the control to be transferred

→ label : statements ;

Rules :-

- Valid C identifier
- A colon should follow label
- No need to declare label

Example :

```
goto l1;
- - -
- - -
l1 : - - -
```

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: __/__/____
PAGE: __/__/____

Looping Statements :-

- To execute a group of statements repeatedly until some condition satisfied.
- Types
  i. while
  ii. do--- while
  iii for

i. while structure :-

- simple
- General form,

```
while (test condition)
  {
    body of the loop
  }
next statement;
```

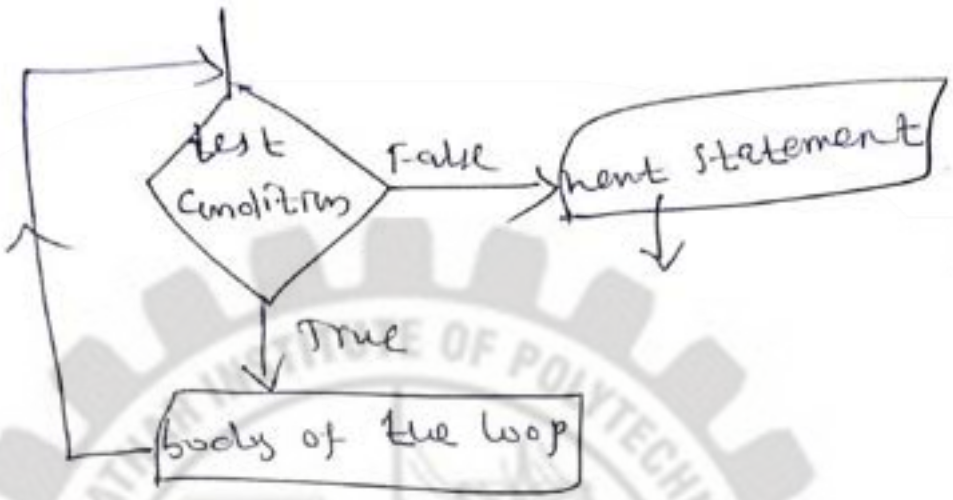
- Evaluates test condition
- If false control to next statement
- If true body of the loop executed until test condition false.
- When false control to next statement

Rules :-

- Test condition, relational or logical expression
- If more than one statement enclosed within brackets.

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Flow diagram:



Example:-

```
#include<stdio.h>
main()
{
  int i;
  i=1;
  while(i<5)
    {
    printf("God loves you");
    i++;
}
}
```

Prints God loves you 4 times

→ i<5 tested.

→ If false body of loop will not be executed

→ If true executed 4 times.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

ii). do ... while

→ The general form,

```
do
{
    body of the loop;
}
while (test condition);
next statement;
```

→ Body of the loop executed first.
→ Then test condition evaluated.
→ If false control to next statement.
→ If true body of the loop executed until the test condition becomes false.

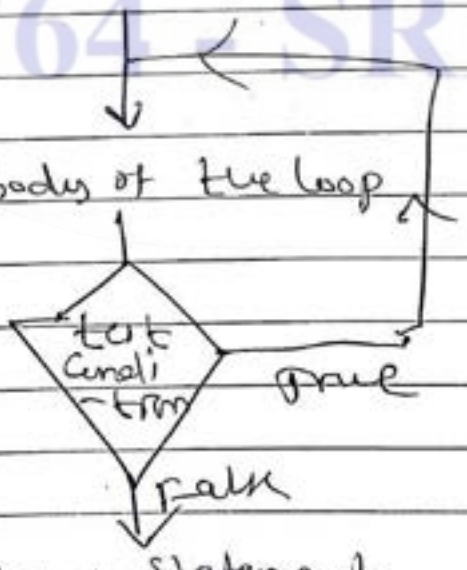
SRIPC ECE

Rules:-

→ Test condition, relational or logical expression
→ If more statements in body of the loop then brackets must.
→ while statement immediately after body of the loop.

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC

Flow diagram:



Protected with free version of Watermarkly. Full version doesn't put this mark.

Example:

```
#include <stdio.h>
main()
{
  int i;
  i=1;
  do
  {
  printf("God Loves You");
    i++;
  }
while(i<5)
}
```

- displays God Loves you 4 times.
- Body of the loop executed
- Condition i<5 tested
- If false program ends
- If true body of loop executed 3 times.

SRIPC ECE

A Difference between while and do-while statement

while	Do-while
1. Entry check	1. Exit check
2. If test condition false body of the loop will not be executed	2. At least once body of the loop executed

REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

iii). for Statement

— To execute a Statement or a group of statement repeatedly for a known number of times.

— The general form,

```
for(Control variable; Test Condition; Increment or decrement)
  {
    body of the loop;
  }
Next Statement;
```

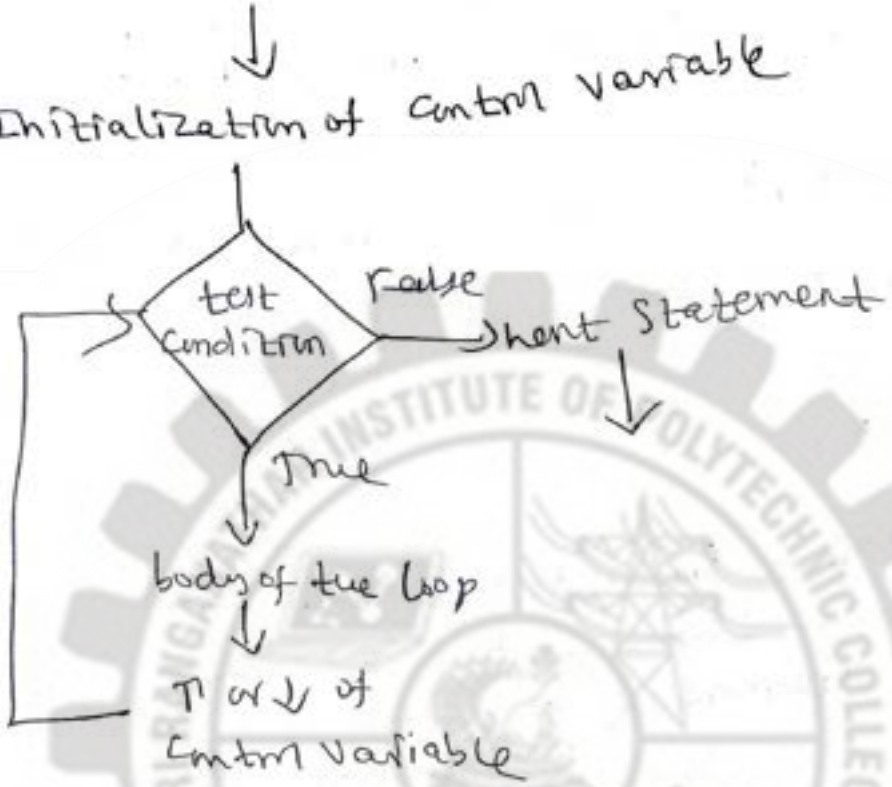
— Control variable - to control the loop
— Test Condition - Check control Variable
— Increment or decrement - Change the value of the control variable

SRIPC ECE

→ When executed,
— Value of control Variable initialized & tested with the test Condition.
— If true body of loop executed
— Control transferred to for Statement
— Value of control Variable ↑ed or ↓ed.
— when test condition false control to Next Statement
— executed repeatedly as long as the test value is true.

Protected with free version of Watermarkly. Full version doesn't put this mark.

Flow diagram:



Example:

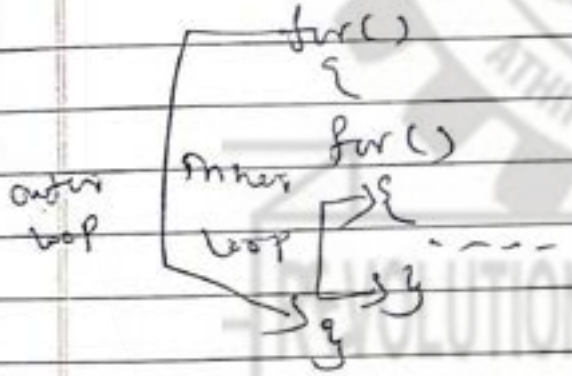
```
#include<stdio.h>
main()
{
 int i,sum;
  sum=0;
 for(i=1;i<=100;i++)
   {
    sum=sum+i;
   }
 printf("sum=%d",sum);
}
```

SRIPC ECE
SRI RAMAKRISHNA INSTITUTE OF POLYTECHNIC COLLEGE
ATHIPALAYAM COIMBATORE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

→ Control variable i initialized to 1 & test condition i<=100 evaluated
→ If true, body of the loop Sum = Sum + i executed & control transferred to for statement
→ value of control variable Incd, i++
→ Checked with test condition i<=100
→ If true body of the loop executed
→ Continues until i<=100 false.
→ when false control to next statement
→ The value of sum printed.

* Nested for loop :
- one for loop enclosed within another for loop
- no limit

Example :



Rules :
1. Running variables should not be the same
2. Jumping from outside loop to inner loop is not allowed
3. Jumping from inside loop to outer loop is allowed

4. The loops should not overlap.

SRIPC ECE
764-SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Break Statement:-

- Used to exit from a loop while the test condition is true
- Used within for, while, do-while and switch statement
- General form,

```
break;
```

- When executed inside a loop, execution terminated & statement following the loop executed
- If used in nested loops, it will exit from the loop containing it.

Example:

```
1. while (condition)
   {
   - - -
   - - -
   if (condition)
   {
   - - -
   break;
   }
   - - -
   }
   Next Statement
```

SRIPC ECE
SRM VALLIAMMAI INSTITUTE OF POLYTECHNIC COLLEGE ATHIPALAYAM COIMBATORE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE / /
PAGE / /

Continue statement:

- Skip the remaining loop statements and control transferred to beginning of the loop.
- statements after continue skipped and control transferred to beginning of the loop
- The general form,

```
continue;
```

Example:

```
for (i=1; i<=100; i++)
  {
  scanf("%d",&a);
  if(a<0)
    {
    printf("error\n");
    continue;
    }
  ---
  ---
  statements for Processing positive value
  ---
  ---
  }
```

- For negative value of a statements below continue skipped and control transferred to beginning of the loop

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: / /
PAGE: / /

UNIT-III

ARRAYS AND STRINGS

Arrays :-

- In Computers we Store information in the memory by giving names to them.
- names called Variables

Scanf("%d%d%d", &a, &b, &c);

If a, b, c values are 10, 20, 30, they stored in memory cells as

a	b	c
10	20	30

- If we want to store 1000 numbers for processing it is difficult to name 1000 different variables
- with the help of arrays, 1000 numbers called as array of numbers
- Single name to all 1000 numbers
- If a is the name given to all 1000 numbers, first number referred as a[0] and second as a[1] and so on.
- The individual variables a[0], a[1] ... a[999] called Subscripted Variables or index variables.
- the numbers are stored in the memory as

a[0]	a[1]	a[2]	...	a[999]
10	20	30		1999

Array definition:-

- Group of related data items stored by means of a single variable name.

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

→ The variables used to represent the individual element or data in the memory called Subscripted variables.

→ General form,

array-name [subscript]

→ array-name - valid C-variable name

→ subscript - an integer from 0 to n.

SRIPC ECE

Rules:-

- Subscript - integer
- " can not be -ve
- given within square brackets after array name
- more than one subscript, separate square brackets
- Subscripts - integer variables or expressions that give integers.

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ..../..../....
PAGE: ..../..../....

One dimensional array :

- An array name with only one subscript is known as 1-D array
- General form,

array-name [subscript]

Example :

i) a[1]

ii) grade [50]

a & grade are name of the arrays

1 & 50 are subscripts representy data in 1st & 50th location in memory

Declaration :

- Array must be declared before it is used like other variables.
- General form

datatype array-name [size];

Where,

datatype - int, float, char etc.,

array-name - valid C variable name

size - no. of contiguous location in the memory to be reserved

Rules :

1. array name - valid C variable.
2. name of the array unique
3. elements same type.

Example :

i. int mark[100]

- Integer type array
- mark
- having 100 memory locations to store 100 integer data.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

ii) float salary[100];

- Floating point array
- salary handle 100 memory locations to store 100 floating point data

iii). char name[100];

- Character type array
- name handle 100 memory locations to store 100 characters.

Array initialization:-

- Assigning initial values to 1-D arrays

during declaration

Static datatype array_name[size] = {list of values};

Static - keyword

datatype - type of data (int, float, char)

array_name - valid C variable name

size - no. of continuous memory locations

list of values - initial values

i. static int arr[3] = {10, 20, 30};

arr - integer array having 3 locations & assigns initial value as,

arr[0]	arr[1]	arr[2]
10	20	30

SRIPC ECE
REVOLUTION THROUGH HIGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ...../...../.....
PAGE: ...../...../.....

1-D Array processing (Reading and writing)

- Looping statements used
- Storing values in array (Reading)
- Retrieving the stored values (writing)
- For loop used

Example : 1-D array Reading

```
Int a[10];
for(i=0; i<10; i++)
  {
  scanf("%d", &a[i]);
  }
```

- Used to store 10 integer values in an array named a

SRIPC ECE

Ex. 1-D array writing

```
Int a[10];
for(i=0; i<10; i++)
  {
  Printf("%d", a[i]);
  }
```

Table handling

- Horizontal or vertical
- When data to be represented in a horizontal as well as vertical tabular form headed, 2-D arrays represent table in memory.

PROMOTION THROUGH TECHNOLOGY
764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Two dimensional array:-
  → An array with 2 subscripts
  → General form,
  array-name [subscript 1] [subscript 2]
  ↓ row no. ↓ column no.
  → Example a[1][2]
  ↓
  2-D array named a.

* Declaration:-
  - declared like other variables
  - General form,
  datatype array-name [row size] [column size]
  ↓ ↓ ↓ ↓
  int/float/char Valid C variable name max. no of rows max. no of columns

* Rules:-
  - Arrayname - valid C variable name
  - Arrayname unique
  - array elements same type
  - size (row x column)

* Example:
  int mark[5][2];
  mark - 2-D int array, 5 rows & 2 columns
  no. of locations 5x2 = 10

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

	Column 0	Column 1
Row 0	mark[0][0]	mark[0][1]
Row 1	mark[1][0]	mark[1][1]
Row 2	mark[2][0]	mark[2][1]
Row 3	mark[3][0]	mark[3][1]
Row 4	mark[4][0]	mark[4][1]

→ I Subscript or index - row number
→ II " " - Column "
→ mark[0][0] ---- mark[4][1] - address of location.

* Array Initialization :-
- Assigning initial values to 2-D array during declaration
- General form

Static datatype array_name[row size][column size] =
{ list of values };

static - keyword
list of values - initial value separated by commas.

Example :
i. Static int mark[3][2] = { 1, 4, 6, 3, 4, 8 };

mark[0][0]	mark[0][1]
1	4
mark[1][0]	mark[1][1]
6	3
mark[2][0]	mark[2][1]
2	8

ii. Static int mark[3][2] = { {1, 4}, {6, 3}, {4, 8} };

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* 2-D arrays processing (reading and writing)
  → Looping statements used.
  → i. Storing values in arrays (Reading)
  → ii. Retrieving " " (writing)

  → for loop used.
  → For 2-D array 2 for loops used

i. Reading

```
int a[50][50], i, j, m, n;
scanf("%d %d", &n, &m);
for(i=0; i<m; i++)
{
for(j=0; j<n; j++)
{
scanf("%d", &a[i][j]);
```

ii. writing

```
int a[50][50], i, j, m, n;
scanf("%d %d", &m, &n);
for(i=0; i<m; j++)
{
for(j=0; j<n; j++)
{
printf("%d", &a[i][j]);
```

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ...../...../.....
PAGE: ...../...../.....

Multidimensional Arrays :-

- 3 or more dimensions
- limit determined by compiler
- General form,

  datatype arrayname [s1][s2][s3] ... [sn];

- Example

  Int marks [5][5-0][6];

- 150 elements

Strings :-

→ Sequence of 0 or more characters enclosed within double quotes

→ null character \0 at end not visible when displayed

→ Size = no. of char + 1

Declaration :-

→ Character type array

→ Char variablename [size];

datatype — valid C variable name — max. no char including \0

Initialization :-

Static Char variablename [size] = String;

keyword or storageclass

Static char city[10] = "COIMBATORE"

C O I M B A T O R E \0

SRIPC ECE
EVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Reading Strings:-

→ Scanf

→ getchar

→ gets

Scanf() :-

%s

scanf("%s", variablename);

%- conversion specification indicator

s - type specifier

variablename - valid C variable

Problem:-

reading stops when it finds a white space

Char (blanks, tabs)

- possible space.

Ex:

Char name[20];

scanf("%s", name);

Raja Ram

Raja only read into memory

getchar()

read a single character

variablename = getchar();

↓

valid C variable

name

Use function repeatedly → String of character read into memory

SRIPC ECE

REVOLUTION THROUGH TECHNOLOGY

764 SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

iii. Gets()

- read a string of characters until a new line character \n is entered.
- Gets(variablename);

Ex:

```
Char name[10];
gets(name);
```

Raja Ram \n

writing Strings

Printf

Putchar

Puts

Char s[10];

printf("%s", variablename);

Char S

getChar(S);

PutChar(S)

```
Char S[10];
Gets(S);
puts(S);
```

String handling functions:-

String.h

1. Strlen() – no. of Characters in a Given String
2. Strcpy() – Copy content of one string into another string
3. Strcat() – join 2 strings
4. StrCmp() – Compare 2 Strings
5. Strrev() – reverse a string

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
704 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Table of Strings :-

- List of names, Places etc.)
- 2-D array of characters
- Each row separate string

Char variable-name [row size] [column size];

datatype — valid (variable name — no. of entries — max. no of characters

Example :

1. Char name[5][26];

* Null String :-

- Contains no character other than null character
- Char variable-name [ ];

SRIPC ECE

REVOLUTION THROUGH TECHNOLOGY

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ..../..../....
PAGE: ..../..../....

UNIT-IV

Structure and Union

→ Array - represent group of data items of same data type with a single name.

→ Structure " " different " "

Definition :-

→ Data type to represent several diff. types of data with a single name.

Diff. b/w Arrays and Structures :-

Arrays	Structures
1. All data same data type.	Data can be of different data types
2. Individual entries are called elements	Individual entries are called members.

Difference in defining ordinary variable and structure variable

- Integer variable i in C defined as
  Int i;
  C knows what is Int
  but in Structure we have to tell about
  1. how the structure looks like
  2. the variables.

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Need of Structure :-
- refer several data types using single name
- used to implement user defined data types (Linked list, trees, graphs etc.)
- Store information in the form of a record
- basis for objects in object oriented programming

Structure definition :-
- Contains a keyword struct and a user defined tag-field followed by the members of the structure within braces

```
struct tag_field
{
datatype member1;
datatype member2;
- - - -
- - - -
datatype membern;
};
```

struct - keyword to define structure

tag-field - name of the structure - valid name

datatype - valid datatypes such as int, float etc;

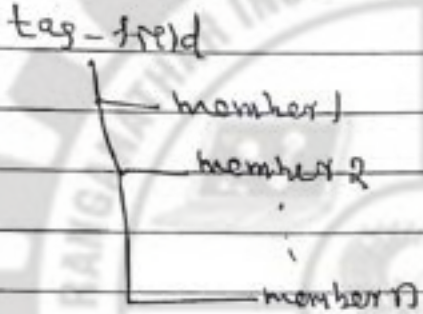
SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE / /
PAGE / /

Rules:-

- Tag-field - name given to the structure
- Each member dec. should be terminated with semicolon
- Compound statements, so it have its own opening and closing braces
- Semicolon after closing brace is must

Struct



Ex:

Stud. Information

Struct Student

{

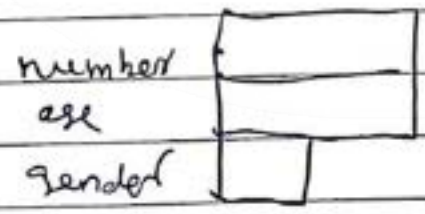
Int number

Int age

Char gender;

};

- name of the structure Student
- 3 members 1. number, age & gender



members do not occupy any memory space

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Structure declaration or variable declaration :-
  - Defining variables to the already defined structure
  - Struct tag_field variable1, variable2, ... variablen;
  - variable1, variable2, ... variablen are valid
    (variable each having n-members
    member1, member2, .. ; membern)
    tag_field - name of the defined structure.

En :

Struct Stu dat Student1, Student2, Student3;

are Structure variables

each having 5 members number are Gender.

SRIPC ECE

* Accessing and giving values to structure members :-
  • Dot operator or member operator . is used to give data to the Structure variables individual members.

Structure variable . member name

The variable name with a Period and the member name is used like any ordinary variable.

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

Ex:

i). Struct student
{
int number;
int age;
char gender;
}student1;

Variable student1 having 3 members.
The data given by,

1. Using keyword
scanf("%d", student1.number);
scanf("%d", student1.age);
scanf("%c", student1.gender);

SRIPC ECE

2. Using assignment statement
student1.number = 101;
student1.age = 21;
student1.gender = 'm';

REVOLUTION THROUGH TECHNOLOGY
704 - SRIPC

Structure initialization:
- Assising initial values
- Static

static struct tag-field structure variable = { value1, value2, ... value n };

static → Storage class
struct → keyword

Protected with free version of Watermarkly. Full version doesn't put this mark.

Example:-

i. struct Student
{
int number;
int age;
char gender;
} static student1 = {100, 21, 'm'};

100 to student1.number
21 to student1.age
m to student1.gender

ii. struct student
{
int number;
int age;
char gender;
};

static struct Student student1 = {100, 21, 'M'};
2 = {101, 21, 'F'};

iii. struct Student
{
int number;
int age;
char sender;
} static Student1 = {100, 21, 'M'};

static struct student student2 = {102, 20, 'F'};

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

iv).

```
Struct Student
  {
  int number ;
  int age ;
  Char gender ;
  } ;
Static Struct Student student[3] = {0,0,0};
```

Initial value will be assigned to first number only
0 to rest of the members

Comparison of Structure Variables:-

Structure variable can be compared as wither other ordinary variable

```
Struct Student
  {
  int number ;
  int age ;
  char gender ;
  } ;
Static Struct Student S1 = {101, 21, 'M'};
"  "  S2 = {101, 21, 'M'};
```

S1, S2 — Structure Variables

S1 = S2 → S2 values assigned to S1

S1 == S2 — compares all the member values
↓ if equal else 0

S1 != S2 → ↓ if not equal else 0

SRIPC ECE
EVOLUTION THROUGH TECHNOLOGY
764 SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Arrays of Structures:-
  - Giving one name to store more than one structure variable

Struct tag field Variable name [size];

keyword — name of the declared structure — number of structure variables to store

Example:

i). Struct Student
{
int number;
int age;
char sender;
};
Struct Student student[2];

student[] — Array of structures having 2 elements
student[0], student[1]
- Each element has 2 members

student[0].number .age .sender

student[1]. " " "

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

```
struct Student
{
int number;
int age;
};

static struct Student Student1[2] = {
                                    {10, 21},
                                    {10, 21}
                                    };
```

Arrays within Structures:
- members of the structure - array data type

Example:

```
struct Student
{
char name[10];
int number;
int age;
char Sender;
int mark[5];
} Student1[5];
```

Student1 - array of structures having 5 elements

- member name Char array - mark. 20 char
- " mark int array - 5 elements
  mark[0], ... mark[4]

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

members accessed as,

student[n].name
n .member
.age
.gender
.mark[0]
.mark[1]
:
mark[4]

* Structures within Structures
[Nested Structure]

— structure is declared as a member of another structure

Structure
member1
member2
:
Structure1
member1
member2
:
Structure n
member1
:
membern

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

```
struct student
{
char name[20];
int number;
char branch[5];
struct hostel_detail
{
int hostel_room_number;
char food;
int deposit amt;
}hostel;
}student1[50];
```

Student1[0].number

→ .hostel.food

SRIPC ECE

Union:

— All data members share the same memory area.

Adv: — Conserve memory space, useful for applications where values not assigned at one time

Union	Structure
Storage for one member at a given time	for all members
memory allocation — member which req. largest memory space	Each member given memory space
Can refer ~~one~~ ensure V in one time	we can refer all the variables.

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: / /
PAGE: / /

UNIT - V

Functions and C files I/O.

Function :-

→ 2 types

→ Inbuilt or Library functions — Function in C programmer need not to write - header file to be add
Ex : Printf, Scanf.

→ User defined functions
↓ or function
written by programmer for particular task

* Inbuilt function :-
→ Already written
→ found in header file
→ To be included in a program.
i. Standard input/output functions
ii. Math functions
iii. Character oriented functions
iv. Graphical functions

SRIPC ECE

User defined functions :
— Group of statements written by programmer
— C program contains one or more functions
— main() one of function (shows where the program execution begins.
— All other functions controlled by main

Need :-
1. Reduced Complexity — Large problem to Small problem
2. Reusability — Can reuse by other user
3. Easy debugging — Error detection and correction easy
4. Extendability — Can extend.

764-SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Function definition :-

→ 2 parts
1. Function header
2. Statement body

→ General form
- Anywerdatatype valid variable name list of arguments) formal / arguments

header
```
function_type function_name(
argument_declaration;  - declaration of formal arguments
{
```

Statement body
```
local variable declaration;  - declaration of variables in the statement body
executable statement 1;
- - - -
- - - -
return(expression);
}
```
keyword (return value) from one function to other function)

Rules :-
→ List of arguments, argument declaration - optional
→ If no list of arguments empty parenthesis must.
→ Expression in return statement optional
→ Parenthesis in return optional.

* Return Statement :-
→ used to return a value to calling function.

return (or) return (expression);

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

→ If no expression act as closing brace & control back to calling function
→ If expression, value returned. Parenthesis optional

Ex:
```
return;
return(a+b);
return a+b;
```

* Declaring function type:-
  - Optional
  - If no function type mentioned it return integer value

Ex:
(i). function-name(list of arguments)
  - returns integer
(ii). int function-name(list of arguments)
  - returns integer
(iii). float function-name(list of arguments)
  - returns float

* Function returning nothing:-
  - void

void function-name(list of arguments)

Ex:
```
abc(i,j)
 int i,j;
 {
  int k;
  k=i+j;
  return(k);
 }
```
abc - name of function & formal parameters i,j
k = i+j
return k to other function

SRIPC ECE
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Calling a function :-
  - by specify name & list of arguments.
  - function-name (list of arguments);
    actual arguments

Rules :-
  - function-name - (called function)
  - List of arguments optional
  - if no arguments empty pair of parenthesis
  - data type should match
  - called fn. returns only one value

* Function declaration :-
  - declaring the function prototype
    datatype function name();

Ex :-

```
#include <stdio.h>
main()
{
 float x,y;
 float abc();
 scanf("%f%f", &x, &y);
 printf("%f", abc(x,y));
}
float abc(i,j)
float i,j;
  {float k;
   k=i+j;
  return(k);
```

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

* Calling a function :-
  - by specify name & list of arguments.
  - function-name (list of arguments);
    actual arguments

Rules :-
  - function-name (called function)
  - List of arguments optional
  - if no arguments empty pair of parenthesis
  - data type should match
  - called fn. returns only one value

* Function declaration :-
  - declaring user defined function in main
    datatype function name();

Ex :-

```
#include <stdio.h>
main()
{
 float x,y;
 float abc();
  scanf("%f%f", &x,&y);
  printf("%f", abc(x,y);
}
float abc(i,j)
float i,j;
  {float k;
    k=i+j;
  return(k);
```

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

DATE: ../../..
PAGE: ../../..

Formal and Actual arguments :-

Formal arguments :-
- Present in function definition (dummy)
- Previous eg.
  (i,j - dummy
  receives value from x,y

Actual arguments :-
- present in calling function

Function call

i) Call by value

ii) Call by reference

SRIPC ECE

Call by value :-
→ the values of actual arguments passed
→ Actual arguments copied to formal arguments.

Eg:

```
#include <stdio.h>
main()
{
int a,b,c;
a=10, b=5,        a    b    c
c=abc(a,b);      10    5   35
printf("%d",c);
}
int abc(i,j)                    k=i+j;
int i,j;          i    j      return k;
{                 10   5      }
c=c+20           35
int k
```

Protected with free version of Watermarkly. Full version doesn't put this mark.

* Category of function:-

i. Function with no arguments and no return value.

ii. " "

Return value

iii. With arguments and with return value.

i. Eg

```
main()
{
  ---
  message();
  ---
}
void message()
{
  ---
  return;
}
```

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

ii). main
{
int a,b;
message();
}
message()
{
int x,y,value;
value = x+y
return(value);
}

SRIPC ECE

iii). main()
{
int a,b;
message(a,b);
}
message(x,y)
int x,y;
{
int value;
value = x+y
return(value);
}

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.

D. C files I/O :

- scanf & printf used for data transfer
- to handle large amount of data using files.

* Defining a file :-

- Defined in stdio.h
- declared as,

FILE * pointer variable;

↓ ↓

datatype pointer to the data type FILE

* Opening a file :-

- Opening a file must open.
- FILE * pointer variable;

Pointer variable = fopen ("filename", "mode");

↓ ↓ ↓

address of the type FILE name of the file

mode

r : for reading only

w : " writing "

a : " appending

r+ : reading & writing

w+ : " "

a+ : " " appending

SRIPC ECE
REVOLUTION THROUGH TECHNOLOGY
704 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Examples:-

i. FILE *a;
a = fopen("mydata", "r");
- file mydata is opened for reading

ii. FILE *b;
b = fopen("test", "w");
- file test is opened for writing

* Closing the file:-
- An opened file must be closed after all operations
fclose(pointer variable);

Eg:
- - -
- - -
FILE *a, *b;
a = fopen("mydata", "r");
b = fopen("mydata", "w");

I/O operations:-
stdio.h
i). getc(). ii). putc() iii) getw()
iv). putw() v. fprintf() vi. fscanf()

i). getc()
c = getc(pointer variable);

ii) putc()
putc(c, pointer variable);

SRIPC ECE
EVOLUTION THROUGH TECHNOLOGY
764 - SRIPC
Protected with free version of Watermarkly. Full version doesn't put this mark.

Examples:-

i. FILE *a;
a = fopen("mydata", "r");
- file mydata is opened for reading

ii. FILE *b;
b = fopen ("test", "w");
- file test is opened for writing

* Closing the file:-
- An opened file must be closed after all operations
fclose (pointer variable);

Ex:

SRIPC ECE

FILE *a, *b;
a = fopen ("mydata", "r");
b = fopen ("mydata", "w");

I/O operations:-
Stdio.h
i). Getc(). ii). Putc() iii) Getw()
iv) Putw() v. fprintf() vi. fscanf()

i). Getc()
c = Getc (pointer variable);

ii) Putc()
putc (c, pointer variable);

Protected with free version of Watermarkly. Full version doesn't put this mark.

iii). getw()

getw(Pointer variable);

iv) putw()

putw(c, Pointer variable);

v) fscanf()

fscanf(Pointer variable, "control string", list);

vi) fprintf()

fprintf(Pointer variable, "control string", list);

* Error handling :-

- Error during I/O operating
- Error handling functions

i). feof() ii). ferror()

SRIPC ECE

REVOLUTION THROUGH TECHNOLOGY

764 - SRIPC

Protected with free version of Watermarkly. Full version doesn't put this mark.