CHARUTAR VIDYA MANDAL'S
SEMCOM
Vallabh Vidyanagar

Faculty Name: Ami D. Trivedi
Class: FYBCA
Subject: US01CBCA01 (Fundamentals of Computer Programming Using C)

*UNIT – 2 (Basics of Programming)

PROBLEMS ANALYSIS

Before we think of a solution of the problem, we must fully understand the problem. We should also understand that what we want from program.

We must carefully decide:

1. What kind of data will go in program?
2. What kind of output is needed? And
3. What are the conditions which are to be considered?

FORMAT OF SIMPLE C PROGRAMS

```
main()      <-------------  Function name
{           <-------------  Start of program
        .......
        .......   <-------  Program statements
        .......
}           <-------------  End of program
```

e.g.

```
main()
{
    /*……print begins…….*/
        printf("I see, I remember");
  /*……printing ends…….*/
}
```

1. The first line informs that the name of program is main and execution begins at this line. The main() is a special function used by C system to tell the computer that from where the program start. Every program must have exactly one main function.
2. The empty pair of parenthesis after main indicates hat main function has no arguments.
3. The opening brace "{" in second line indicates beginning of main function and closing brace "}" in last line indicates end of function.
4. All the statements between these two lines form the function body. Function body contains set of instructions.
5. In this example, function body contains 3 statements. Out of 3 only printf line is executable statement.
6. Lines beginning with /* and ending with */ are known as comment lines. They are used to increase readability and understanding. They are not executable statements.

GENERAL / BASIC STRUCTURE OF C PROGRAMS

A C program can be viewed as a group of building blocks called functions. A function is a subroutine that may include one or more statements designed to perform a specific task. To write a C program, we first create functions and then put them together.

A C program may contain one or more sections shown in Fig-1.

SEMCOM

Documentation Section
Link Section
Definition Section
Global Declaration Section
main ( ) **Function Section**
{
Declaration part
Executable part
}
Subprogram section

Function 1
Function 2
--- (User-defined functions)

Function n

Fig. 1 An overview of a C program

1. The documentation section consists of a set of comment lines giving the name of the program. the author and other details, which the programmer would like to use later.
2. The link section provides instructions to the compiler to link functions from the system library.
3. The definition section defines all symbolic constants.
4. There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.
5. Every C program must have one main() function section. This section contains two parts, declaration part and executable part.

   The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and the closing braces.

   The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is the logical end of the program. All statements in the declaration and executable parts end with a semicolon.
6. The subprogram section contains all the user-defined functions that are called in the main function. User-defined functions are generally placed immediately after the main function, although they may appear in any order.

All sections, except the main function section may be absent when they are not required.

VARIABLES

Definition: A variable is a data name that may be used to store a data value.

Constants remain unchanged during the execution of a program but a variable may take different values at different times during execution.

A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. E.g. Average, height.

Variable naming rules:

1. Variable names may consist of letters, digits, and the underscore(_) character.
2. They must begin with a letter. Some systems permit underscore as the first character.
3. ANSI standard recognizes a length of 31 character. However, the length should not be normally more than eight characters, since only the first eight characters are treated as significant by many compilers.

4. Uppercase and lowercase are significant. That is, the variable Total is not the same as total or TOTAL.
5. The variable name should not be a keyword.
6. White space is not allowed.

Some examples of valid variable names are: John, Value, T_raise.
Invalid examples include: 123, (area), %, 25th.

If only the first eight characters are recognized by a compiler, then the two names average_height & average_weight mean the same thing to the computer.

Such names can be rewritten as avg_height and avg_weight or ht_average and wt_average without changing their meanings.

DECLARATION OF VARIABLES

After designing suitable variable names, we must declare them to compiler. Declaration does two things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

Primary type declaration

A variable can be used to store a value of any data type. That is, the name has nothing to do with its type. The syntax for declaring a variable is as follows:

data-type v1,v2,….vn;

v1,v2,…vn are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For e.g. int count;

Declaration of variables is usually done immediately after the opening brace of the program. The variables can also be declared outside(either before or after) the main function.

Declaration of variables:

```
float x,y;
int code;
double deviation;
char c;
```

User-Defined Type Declaration

C supports a feature known as “type definition” that allows users to define an identifier that would represent an existing data type.

The user-defined data type identifier can later be used to declare variables. It takes the general form:

typedef type identifier;

Where type refers to an existing data type and “identifier” refers to the “new” name given to the data type. The existing data type may belong to any class of type, including the user-defined ones.

Remember that the new type is ‘new’ only in name, but not the data type. typedef cannot create a new type. Some examples of type definition are:

```
typedef int units;
typedef float marks;
```

Here, units symbolizes int and marks symbolizes float. They can be later used to declare variables as follows:

```
units batch1, batch2;
marks name1[50], name2[50];
```

batch1 and batch2 are declared as int variable and name1[50] and name2[50] are declared as 50 element floating point array variables.

The main advantage to typedef is that we can create meaningful data type names for increasing the readability of the program.

ASSIGNING VALUES TO VARIABLES

Assignment Statement (Assignment operator)

Values can be assigned to variables using the assignment operator = as follows:

variable_name = constant;

Example: **initial_value = 0;**

C permits multiple assignments in one line. For example

intial_value = 0;
final_value = 100; are valid statements.

An assignment statement implies that the value of the variable on the left of the 'equal sign' is set equal to the value of the quantity (or the expression) on the right.

The statement **year = year + 1;** means that the 'new value' of year is equal to the 'old value' of year plus 1.

During assignment operation, C converts the right-hand side to the type on the left. This may involve truncation when real value is converted to an integer.

It is also possible to assign a value to a variable at the time the variable is declared. This takes the following form:

data-type variable_name = constant;

Example: **int final_value = 100;**

The process of giving initial values to variables is called initialization. C permits the initialization of more than one variables in one statement using multiple assignment operators. For example
p = q = s = 0; are valid. The statement initializes the variables p, q, and s to zero.

Remember that external and static variables are initialized to zero by default.

ARITHMETIC EXPRESSION

An arithmetic expression is a combination of variables, constants and operators which are arranged as per the syntax of the language.

Arithmetical expression	C expression
(m+n) (x+y)	(m+n) * (x+y)
$3x^2 + 2x + 1$	3 * x * x + 2 * x + 1
$\frac{ab}{c}$	a * b / c

EVALUATION OF EXPRESSION

Expressions are evaluated using an assignment statement of the form

variable = expression; variable is any valid C variable name.

Examples of evaluation statements are:

x = a * b – c; **y = b / c * a;** **z = a – b / c + d;**

When such statements are encountered then the expression is evaluated first and the result is stored in the left-hand side variable. Previous value of left-hand side variable is replaced by result.

SEMCOM

PRECEDENCE OF ARITHMETIC OPERATORS

An arithmetic expression without parentheses will be evaluated from **left to right** using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in C:

High priority * / %	Low priority + -

The basic evaluation procedure includes 'two' left-to-right passes through the expression. During the first pass, the high priority operators (if any) are applied as they are encountered. During second pass, the low priority operators (if any) are applied as they are encountered.

Consider evaluation statement: **x = a – b / 3 + c * 2 – 1** when a = 9, b = 12, and c = 3, the statement becomes x = 9 – 12 / 3 + 3 * 2 - 1 and is evaluated as follows:

First pass
Step1: x = 9 – 4 + 3 * 2 - 1
Step2: x = 9 – 4 + 6 - 1

Second pass
Step3: x = 5 + 6 - 1
Step4: x = 11 -1
Step5: x = 10

Rules for evaluation of expression

1. First evaluate parenthesized sub expression from left to right.
2. If parentheses are nested, the evaluation begins with innermost sub-expression.
3. The precedence rule is applied in determining the order of operators in evaluating sub-expression.
4. Associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
5. Arithmetic expressions are evaluated from left to right using rule of precedence.
6. When parentheses are used then the expressions within parentheses have highest priority.

DATA TYPES

C supports three classes of data types.

1. Primary (or fundamental) data types
2. Derived data types
3. User-defined data types

All a compilers support five **fundamental data types**, namely integer (int), character (char), floating point (float), double-precision floating point (double) and void.

Derived data types include arrays, functions, structures and pointers.

User-defined data types can be created in C using “type definition” (refer typedef) that allows user to define an identifier that would represent an existing data type.

The user-define data type identifier can be used later on to declare variables. Another user-defined data type is enumerated data type provided by ANSI standard.

1. Primary data types in C

Integer		Character	Float	Void
Signed	**Unsigned type**	char	float	
int	unsigned int	signed char	double	
short int	unsigned short int	unsigned char	long double	
long int	unsigned long int			

SEMCOM

Integer Types

Integers are whole number with a range of values supported by a particular machine.

Generally, integers occupy one word of storage. The size of an integer that can be stored depends on the computer because word sizes of machines may vary.

If we use a 16 bit word length (2 bytes), the size of the integer value is limited to the range -32768 to +32767 (that is, -2^{15} to $+2^{15}-1$).

Size and range of data types on a 16-bit machine

Type	Size(bit)	Range	Format Specifier
int or signed int	16	-32,768 to 32767	%d
unsigned int	16	0 to 65535	%u
short int or signed short int	8	-128 to 127	
unsigned short int	8	0 to 255	
long int or signed long int	32	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	32	0 to 4,294,967,295	

We declare long and unsigned to increase the range of values. E.g. range of unsigned integer numbers will be from 0 to 65535. Unsigned integers are always positive.

Floating point types

Floating point numbers are defined in C by the keyword float.

Floating point (or real) numbers are stored in 32 bits (on all 16 bit machines), with 6 digits of precision.

The type double can be used when the accuracy provided by a float number is not sufficient. A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers.

Double type represents the same data type that float represents, but with a greater precision. To extend the precision further, we may use long double which uses 80 bits.

Size and range of data types on a 16-bit machine

Type	Size(bit)	Range	Format Specifier
float	32	3.4E-38 to 3.4E+38	%f
double	64	1.7E-308 to 1.7E+308	%lf
long double	80	3.4E-4932 to 1.1E+4932	%Lf

Character types

A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage.

The qualifier signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 and 255, signed chars have values from –128 to 127.

Size and range of data types on a 16-bit machine

Type	Size(bit)	Range	Format Specifier
char or signed char	8	-128 to 127	%c
unsigned char	8	0 to 255	

Void types

The void type has no values. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

SEMCOM

OPERATORS

C operators can be classified into a number of categories. They include:

1. Arithmetic operators.
2. Relational operators.
3. Logical operators.
4. Assignment operators.
5. Increment and decrement operators.
6. Conditional operators.

1. Arithmetic operators

Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Integer Arithmetic:

If a and b are integers, then for a = 14 and b = 4 we have the following results:

a – b = 10 a + b = 18 a * b = 56
a / b = 3 (decimal part truncated) a % b = 2 (remainder of division)

If one of them is negative, the direction of truncation is implementation dependent.

That is, 6/7 = 0 and –6/-7 = 0 but –6/7 may be zero or –1.(Machine dependent)

Similarly, during modulo division, the sign of the result is always the sign of the first operand (the dividend). That is,

-14 % 3 = -2 -14 % -3 = -2 14 % -3 = 2

Real Arithmetic:

An arithmetic operation involving only real operands is called real arithmetic. If x, y, and z are floats, then we will have:

x = 6.0 / 7.0 = 0.857143 y = 1.0 / 3.0 = 0.333333 z = -2.0 / 3.0 = -0.666667

Note: The operator % cannot be used with real operands.

Mixed-mode Arithmetic:

When one of the operands is real and the other is integer, the expression is called a mixed-mode expression. Thus 15 / 10.0 = 1.5 where as 15 / 10 = 1.

2. Relational operators

We often compare two quantities and depending on their relation we take certain decisions.

For e.g. we compare age of two persons or price of two items. These comparisons can be done with the help of relational operators.

An expression such as a < b, 1>20 which contains a relational operator is known as relational expression.

Value of relational expression is either 1 or 0. If relation is true then value is 1 and if relation is false then value is 0.

e.g. 10 <= 10 is TRUE 10 > 20+4 is FALSE

SEMCOM

C supports 6 relational operators.

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Syntax of simple relational expression:

ae-1 relational operator ae-2

where ae-1 and ae-2 are arithmetic expressions which may be simple constants, variables or combination of them.

3. Logical operators

In addition to the relational operators, C has the following three logical operators.

Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

The logical operators && and || are used when we want to test more than one condition and make decisions. Example: **a > b && x == 10**

An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression.

Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table.

The logical expression given above is true only if a > b is true and x == 10 is true. If either (or both) of them are false, the expression is false.

Truth table

op - 1	op - 2	Value of the expression	
		op - 1 && op - 2	op - 1 || op - 2
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Some examples of the usage of logical expressions are:

1. if (age > 55 && salary < 1000)
2. if (number < 0 || number > 100)

4. Assignment operators

Assignment operators are used to assign the result of an expression to a variable. We use assignment operator '=' for this purpose. (Refer Assignment statement page-5)

C also has set of **shorthand assignment operators**.

Syntax: v op = exp;

where v is a variable, exp is an expression and op is a C binary arithmetic operator.

The operator **op=** is known as shorthand assignment operator.

The assignment statement **v op = exp;** is equivalent to **v = v op (exp);**

e.g. **a *= 2; is same as a = a * 2;**
x += y+1; is same as x = x + (y+1);
c /= 3; is same as c = c / 3;

Shorthand Assignment Operators:

Statement with simple assignment operator	Statement with shorthand operator
a = a + 1	a += 1
a = a - 1	a -= 1
a = a * (n+1)	a *= n+1
a = a / (n+1)	a /= n+1
a = a % b	a %= b

Advantages of Shorthand assignment operators:

1. Left-hand side variables need not be repeated. So it is easy to write.
2. Statement is more short and easy to read.
3. Statement is more efficient.

5. Increment and decrement operators

These are the increment and decrement operators: ++ and –

The operator ++ adds 1 to the operand while – subtracts 1. Both are unary operators and take the following forms:

++m; or m++;
--m; or m--;

++m; is equivalent to **m = m + 1**; (or **m +=1;**)
--m; is equivalent to **m = m – 1;** (or **m - =1**;)

While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement.

Consider the following:

Case 1:	**Case 2:**
m = 5; **y = ++m;** In this case, the value of y and m would be 6.	**m = 5;** **y = m++;** Here, the value of y would be 5 and m would be 6.

So, a **prefix operator** first adds 1 to the operand and then the result is assigned to the variable on left. On other hand, a **postfix operator** first assigns the value to the variable on left and then increments the operand.

6. Conditional Operator (Ternary Operator)

A ternary operator pair "**? :**" is available in C to construct conditional expressions. This operator is useful fro making two-way decisions. This operator is popularly known as the conditional operator.

The general from is: **exp1 ? exp2 : exp3;**

where exp1, exp2 and exp3 are expressions.

The operator **? :** works as follows:

exp1 is evaluated first.

If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression.

SEMCOM

If exp1 is false, exp3 is evaluated and its value becomes the value of the expression. Note that only one of the expressions (either exp2 or exp3) is evaluated.

For example,

```
a = 10;
b = 15;
x = (a>b) ? a : b;
```

In this example, x will be assigned the value of b. This can be achieved using the if..else statements as follows:

```
if (a>b)
        x = a;
else
        x = b;
```

READING DATA FROM KEYBOARD – scanf()

Values can be assigned to the variable by 2 ways:

1. using assignment operator = and
2. input data through keyboard using the **scanf** function

The general format of scanf is as follows:

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

Control string specifies the field format in which the data is to be entered and the variables variable1, variable2… specify the address of locations where the data is stored. Control string and variables are separated by commas.

Control string (also known as format string) contains field specifications which gives explanation of input data. It may include:

1. Field (or format) specifications which consist of conversion character %, a data type character (or type specifier), and an optional number which specify field width.
2. Blanks, tables or newline.

Blanks, tables or newline are ignored. The data type character indicates the type of the data to be assigned to the variable associated with corresponding variable. The field width specifier is optional.

The ampersand symbol & before each variable name is an operator that specifies the variable name's address. We must always use this operator, otherwise unexpected results may occur.

Example: **scanf("%d",&number);**

When this statement is encountered by the computer, the execution stops and waits for the value of the variable number to be typed in.

Since the control string "%d" specifies that the integer value is to be read from the terminal, we have to type in the value in integer form. Once the number is typed in and the 'Return' key is pressed, the computer then proceeds to the next statement.

Thus, the use of scanf provides an interactive feature and makes the program 'user friendly'.

PRINTING DATA – printf()

printf function is used for printing messages and numerical results. Outputs are to be produced in such a way that they are understandable. So appearance and clarity of output of program is to given careful consideration.

The general purpose of printf statement is to print the message.

Example: **printf ("Hello Everybody");**

The general format of printf is as follows:

printf ("control string", variable1, variable2,….);

The control string contains the format of data being received. Control string consists of 3 types of items:

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display each item.
3. Escape sequence characters such as \n, \t etc.

Example: **printf ("%d",number);**

When this statement is encountered by the computer, the computer prints the value of the variable that is read by the user.

Since the control string "%d" specifies that the integer value is read from the terminal, we will get the value in integer form.

Once the number is printed on the monitor, the computer then proceeds to the next statement.

Control string indicates number of arguments (variables) along with their data types. variable1, variable2 … are the variables whose values are formatted and printed according to the specification of the control string.

Variables should match in number, order and type with format specifications.

A simple format specification has the following form:

% w.p type-specifier

Where w is an integer number that specifies the total number of columns for the output value and p is another integer number that specifies the number of digits to the right of the decimal point (of a real number) or the number of characters to be printed from a string. Both w and p are optional.

DECISION MAKING STATEMENTS / CONTROL STRATEGIES

In practice, we have a number of situations where we may have to change the order of execution of statements based on certain conditions or repeat a group of statements until certain specified conditions are met.

C language possesses such decision making capabilities and supports the following statements known as control or decision making statements.

1. if statement
2. switch statement
3. Conditional operator statement
4. goto statement

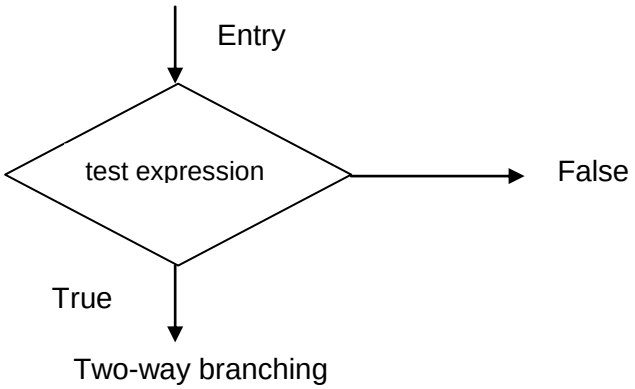
1. IF STATEMENT

Syntax: if (test expression)

It allows the computer to evaluate the expression first.

And then, depending on whether the value of the expression (relation or condition) is 'true' (non-zero) or 'false' (zero), it transfers the control to a particular statement.

This point of program has two paths to follow, one for the true condition and the other for the false condition and the other for the false condition.



Two-way branching

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

(1) Simple if statement
(2) if….else statement
(3) Nested if….else statement
(4) else if ladder

(1) Simple IF statement

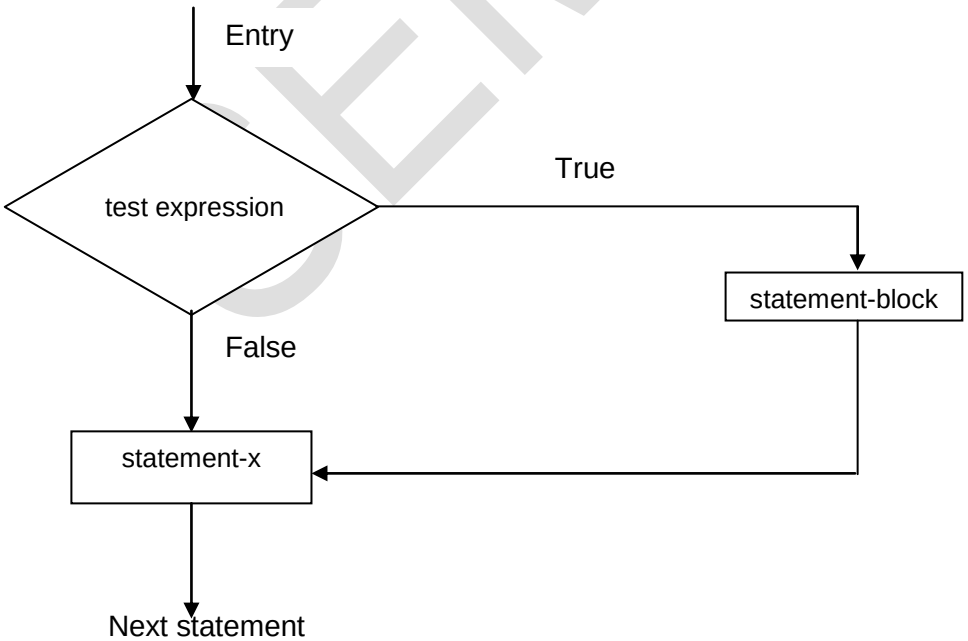
The general form of a simple if statement is

```
if (test expression)
  {
     statement-block;
  }
statement-x;
```

The 'statement-block' may be a single statement or a group of statements.

If the test expression is true, the statement-block will be executed; otherwise the statement-block will be skipped and the execution will jump to the statement-x.

Flowchart of simple if control



e.g. if (a > b)
printf ("a is greater than b");

(2) IF…ELSE statement

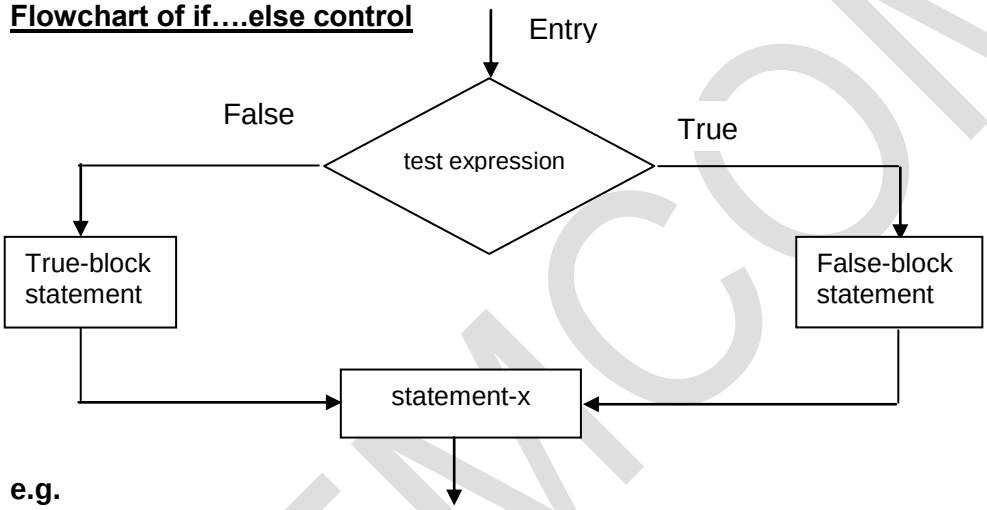
The if…else statement is an extension of the simple if statement. The general form is

```
if  (test expression)
    {
        true-block statement(s)
    }
else
    {
        False-block statement(s)
    }
statement-x
```

If the test expression is true, then the true-block statement(s) (i.e. immediately following the if statements) are executed. Otherwise the false-block statements are executed.

Either true-block or false-block will be executed, not both. In both the cases, the control is transferred to statement-x.

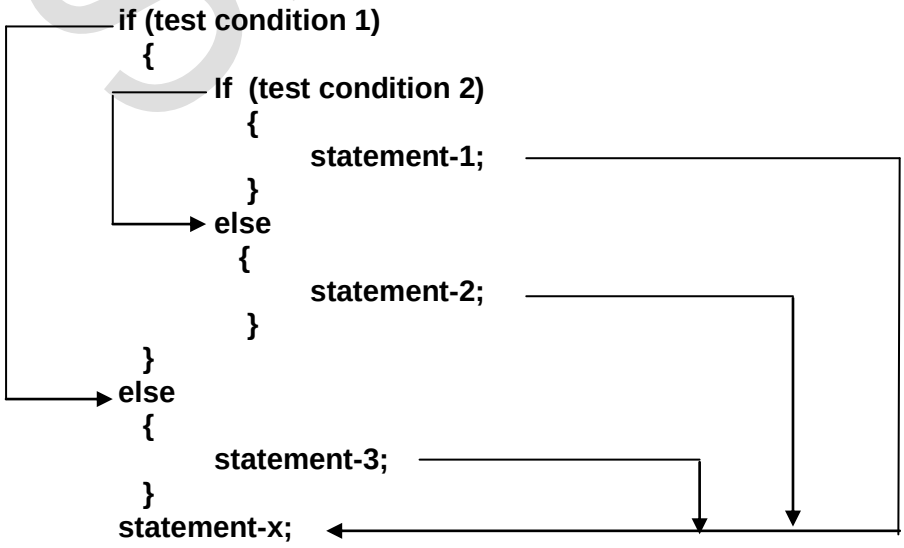
Flowchart of if….else control



e.g.

```
if (code = = 1)
        bonus = salary * 10 / 100;
else
        bonus = salary * 20 / 100;
```

(3) Nesting of IF…ELSE statement (Nested if)



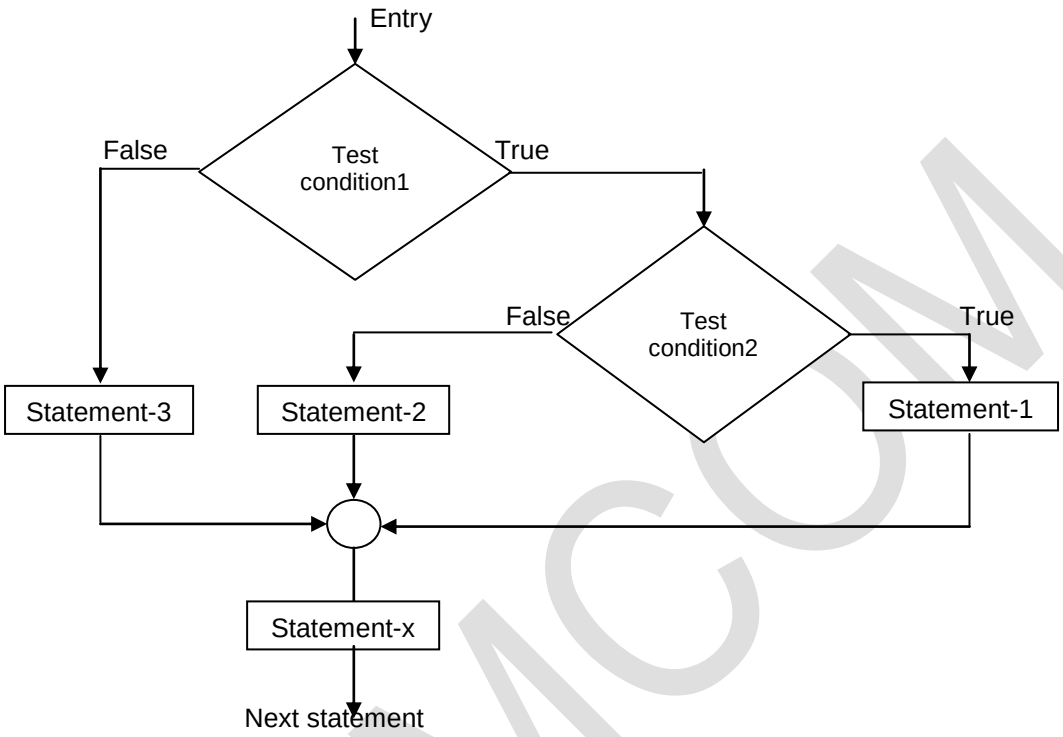
SEMCOM

When a series of decisions are involved, we may have to use more than one if…else statement in nested form. The logic of execution is illustrated in the flowchart.

If the condition-1 is false, the statement-3 will be executed; otherwise it continues to perform the second test.

If the condition-2 is true, the statement-1 will be evaluated; otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

Flowchart of nested if…else statement



e.g.

```
if (code = = 1)
  bonus = salary * 10 / 100;
else
      if (code = = 2)
         bonus = salary * 20 / 100;
      else
           if (code = = 3)
             bonus = salary * 30 / 100;
           else
              bonus = salary * 40 / 100;
```

2. switch statement

When one of the many alternatives is to be selected, we can use an if statement.

But complexity of such program increases when number of alternatives increases. The program becomes difficult to read and follow.

C has a built-in multiway decision making statement known as a switch.

Switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statement associated with that case is executed.

SEMCOM

Syntax:

```
switch (expression)
 {
        case value-1:
                block-1
                break;
        case value-2:
                block-2
                break;
        …….
        …….
        default:
                default-block
                break;
 }
statement-x;
```

The expression is an integer expression or characters.

value-1, value-2…. Are constants or constant expressions (evaluated to an integral constant) and they are known as case labels. Each of these values should be unique within switch statement. Case label ends with **:**.

block-1, block-2…. are statement lists and may contain zero or more statements. There is no need to put braces around these blocks.

When switch statement is executed, the value of the expression is compared against the values value-1, value-2,…. If a case is found whose value matches with the value of the expression, then the block of statements that follows the case is executed.

The break statement is used at the end of each block to indicate the end of a particular case. It causes an exit from the switch statement and transfers the control to statement-x following the switch.

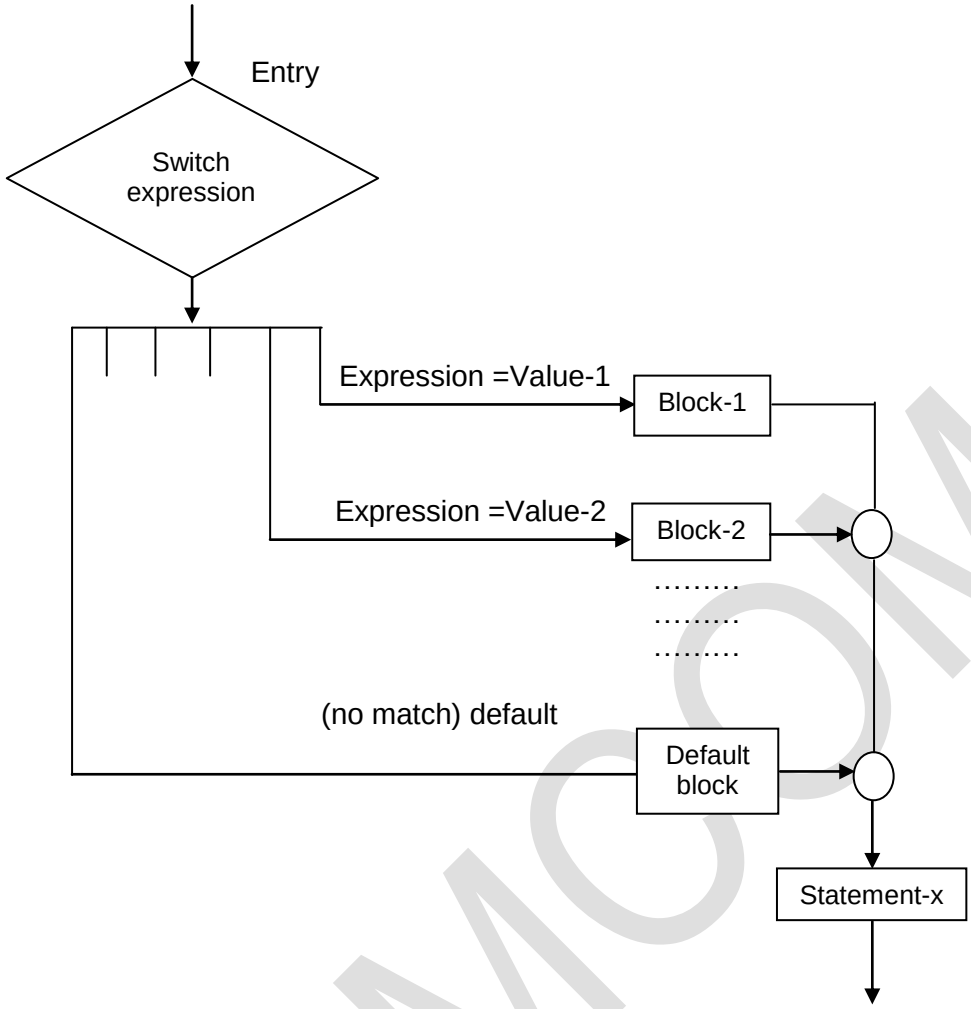
The default is an optional case. When it is present, it will be executed if the value of the expression does not match with any of the case values. If it is not present, no action takes place if all matches fail and the control goes to the statement-x.

e.g.

```
scanf ("%d  %d",&salary,&code);
switch (code)
{
        case 1:
        case 2:
        case 3:   bonus = salary * 10 / 100;
                  break;
        case 4:
        case 5:   bonus = salary * 20 / 100;
                  break;
        case 6:
        case 7:  bonus = salary * 30 / 100;
                  break;
        default:  bonus = salary * 40 / 100;
                  break;
}
printf("%d",bonus);
```

SEMCOM

Selection process of the switch statement



Disclaimer: The study material is compiled by Ami D. Trivedi. The basic objective of this material is to supplement teaching and discussion in the classroom in the subject. Students are required to go for extra reading in the subject through library work.

SEMCOM