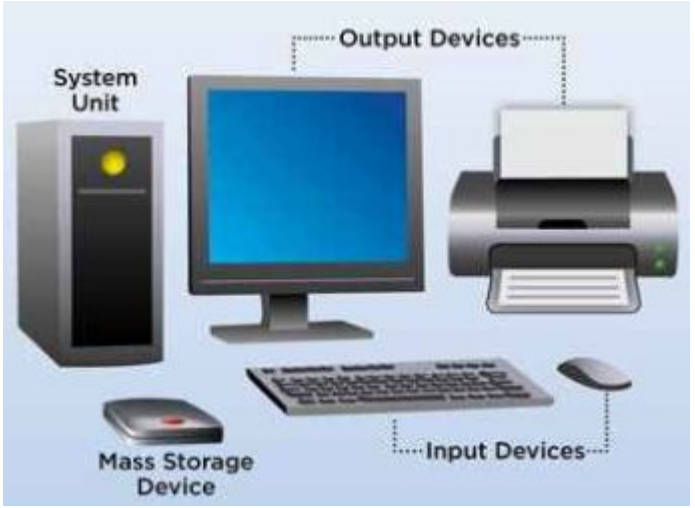
UNIT – II

SLO-1 :
Relational and Logical Operators
SLO – 2 :
Condition Operators, Operator Precedence

Definition : Operator

- Definition “An operator is a symbol (+,-,*,/) that directs the computer to perform certain mathematical or logical manipulations and is usually used to manipulate data and variables” Ex: a+b



Operators in C

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

Arithmetic Operators

Operator	example	Meaning
+	a + b	Addition –unary
-	a – b	Subtraction- unary
*	a * b	Multiplication
/	a / b	Division
%	a % b	Modulo division- remainder

Relational Operators

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Equal to
!=	Not equal to

Logical Operators

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

- Logical expression or a compound relational expression-
- An expression that combines two or more relational expressions
- Ex: if (a==b && b==c)

Truth Table

a	b	Value of the expression	
		a && b	a || b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Assignment operators

Syntax:

v op = exp;

Where, v = variable,

op = shorthand assignment operator

exp = expression

Ex: x=x+3

x+=3

Shorthand Assignment operators

Simple assignment operator	Shorthand operator
a = a+1	a + =1
a = a-1	a - =1
a = a* (m+n)	a * = m+n
a = a / (m+n)	a / = m+n
a = a %b	a %=b

Conditional operators

Syntax:

exp1 ? exp2 : exp3

Where,

exp1,exp2 and exp3 are expressions

Working of the ? Operator:

- Exp1 is evaluated first, if it is nonzero(1/true) then the expression2 is evaluated and this becomes the value of the expression,
- If exp1 is false(0/zero) exp3 is evaluated and its value becomes the value of the expression

Ex:

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

Bitwise operators

- These operators allow manipulation of data at the bit level

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

Bitwise operators

- In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level.
- To perform bit-level operations in C programming, bitwise operators are used.

Bitwise AND operator &

- The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Example : Bitwise AND

Let us suppose the bitwise AND operation of two integers 12 and 25.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

```
  00001100
& 00011001
__________
  00001000  = 8   (In decimal)
```

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Output

Output = 8

Bitwise OR operator |

- The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1.
- In C Programming, bitwise OR operator is denoted by |

Example : Bitwise OR

12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

```
  00001100
| 00011001
 ________
 00011101  = 29 (In decimal)
```

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a|b);
    return 0;
}
```

Output

Output = 29

Special operators

1. Comma operator (,)
2. sizeof operator – sizeof( )
3. Pointer operators – (& and *)
4. Member selection operators – (. and ->)

Operator Precedence

- An expression is evaluated from left to right.
- In C language, each operator has a “precedence” or priority associated with it.
- The operators at the higher level of priority are evaluated first.
- The operators of the same priority are evaluated either from left to right or from right to left, depending on the level. This is known as the associativity property of an operator.
- Example:

(3 * 5) + (4 / 2) – (1 * 2)

15 + 2 - 2

15

Operator Precedence

OPERATOR	DESCRIPTION	ASSOCIATIVITY	RANK
() []	Function call Array element reference	Left to right	1
+ - ++ -- ! ~ * & sizeof (type)	Unary plus Unary minus Increment Decrement Logical negation Ones complement Pointer reference Address Size of an object Type cast		2
* / %	Multiplication Division Modulus	Left to right	3
+ -	Addition Subtraction	Left to right	4
<< >>	Left shift Right shift	Left to right	5
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to right	6

SLO-1 :
Expressions with pre / post increment operator

Pre / Post increment operator

- Increment and decrement operators are unary operators.
- They can add or subtract ‘1’ from the operand.

  Eg:x = x +1 can be written as ++x

  result : x=2

- C languages has two types (pre and post) of each operator:
  1. Operator placed before variable (Pre)
  2. Operator placed after variable (Post)

- The increment/decrement operator is ++ and --.

Increment and Decrement Operators

Types:

1. Pre Increment Operator
2. Post Increment Operator
3. Pre Decrement Operator
4. Post Decrement Operator

Syntax:

(pre)++var_name; (pre)--var_name;

(Or)

var_name++ (post); var_name -- (Post);

Examples:

++a, --total, a--, i--

Increment and Decrement Operators

S. No	Operator type	Operator	Description
1	Pre Increment	++x	Value of x is incremented by 1 before assigning it to variable x.
2	Post Increment	x++	Value of x is incremented by 1 after assigning it to variable x.
3	Pre Decrement	--x	Value of x is decremented by 1 before assigning it to variable x.
4	Post Decrement	x --	Value of x is decremented by 1 after assigning it to variable x.

Increment and Decrement Operators

/* Program for Post Increment*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
     int x = 0;
    while (x++<3)
    {
        printf("%d,", x);
        }
        getch ( );
        }
```

Output

1, 2, 3,

Increment and Decrement Operators

Step 1 : In the program, value of x "0" is compared with 3 in while statement.

Step 2 : Then, value of "x" is incremented from 0 to 1 using post- increment operator.

Step3: Then,this incremented value "1" is assigned to the variable "x".

Above 3 steps are continued until while statement becomes ' false ' and output is displayed as "1, 2, 3".

Increment and Decrement Operators

```
/* Program for Pre Increment*/
#include<stdio.h>
#include<conio.h>
void main()
{
      int x = 0;
     while (++x<3)
     {
          printf("%d,", x);
          }
          getch ( );
}
```

Output

1, 2,

Increment and Decrement Operators

Step 1 : In the program, value of "x" is incremented from 0 to 1 using pre- increment operator.

Step 2 : Then, value of x "0" is compared with 3 in while statement.

Step 3 : Then, this incremented value "1" is assigned to the variable "x".

Above 3 steps are continued until while statement becomes ' false ' and output is displayed as "1, 2".

SLO – 2 :
Expression with conditional and assignment operators

Ternary Operator

- We can also judge the condition using **ternary operator**. Ternary operator checks whether a given condition is true and then evaluates the expressions accordingly. It works as follows.

  condition ? expression1 : expression2;

- If the **condition is true**, then **expression1** gets evaluated, otherwise **expression2** gets evaluated.

```
#include <stdio.h>                    //ternary.C
int main()
{
   int age;
    clrscr();
   printf("Enter age");
   scanf("%d", &age);
   (age > 18) ? printf("eligible to vote\n") : printf("not eligible to vote\n");
    getch();
   return 0;
}
```

Ternary Operator

- Here, if the condition (age > 18) is true, then expression1 i.e. printf("eligible to vote\n") will get evaluated, otherwise expression2 i.e. printf("not eligible to vote\n") will get evaluated.
- Since the value of age that we entered (10) is less than 18, expression2 got evaluated and "not eligible to vote" got printed.

SLO-1 :
if statement in expression

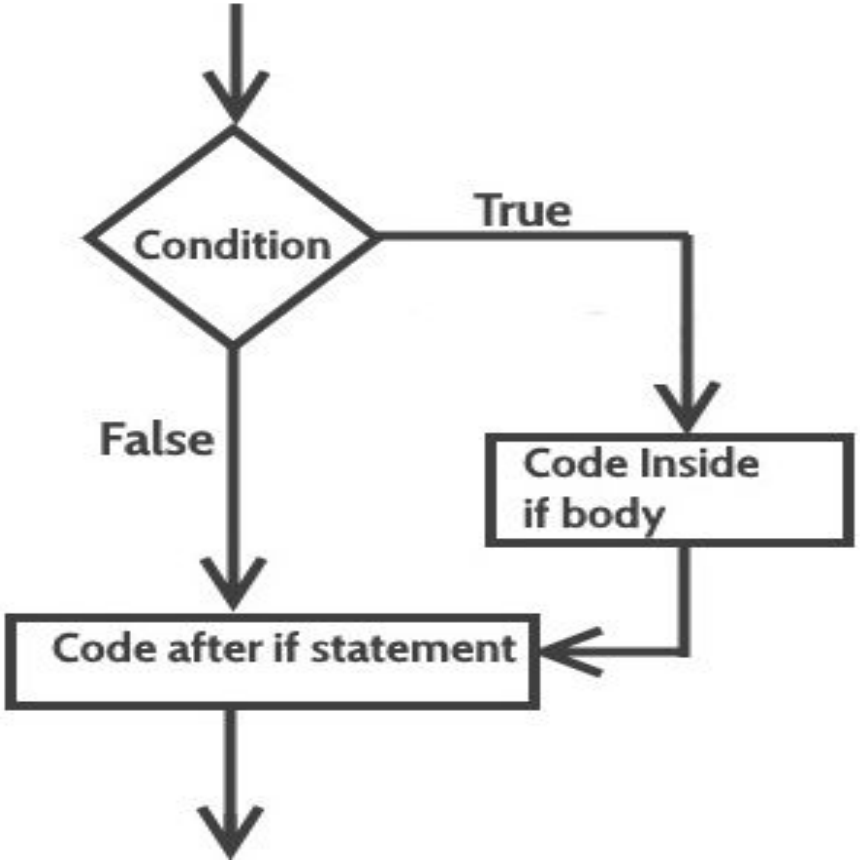
If statement

Syntax of if statement:

The statements inside the body of “if” only execute if the given condition returns true. If the condition returns false then the statements inside “if” are skipped.

```
if (condition)
{
    //Block of C statements here
        //These statements will only execute if the condition is true
}
```

Flow Diagram of if statement



SLO-2 :
L value and R value in expression

Lvalue and Rvalue

In C, an expression is either an lvalue or an rvalue. As you know, every expression has a value. But the value in an expression (after evaluation) can be used in two different ways.

Lvalue and Rvalue

	Expression Type[a]	Comments
1.	identifier	Variable identifier
2.	expression[...]	Array indexing
3.	(expression)	Expression must already be lvalue
4.	*expression	Dereferenced expression
5.	expression.name	Structure selection
6.	expression->name	Structure indirect selection
7.	function call	If function uses *return* by address

[a] Expression types 5, 6, and 7 have not yet been covered.

Table *lvalue* Expressions

Note The right operand of an assignment operator must be an *rvalue* expression.

Lvalue and Rvalue

Type of Expression	Examples
Address operator	`&score`
Postfix increment/decrement	`x++` `y--`
Prefix increment/decrement	`++x` `--y`
Assignment (left operand)	`x = 1` `y += 3`

Table Operators That Require *lvalue* Expressions

Lvalue and Rvalue

Expression	Problem
`a + 2 = 6;`	a + 2 is an rvalue and cannot be the left operand in an assignment; it is a temporary value that does not have an address; no place to store 6.
`&(a + 2);`	a + 2 is an rvalue, and the address operator needs an lvalue; rvalues are temporary values and do not have addresses.
`&4;`	Same as above (4 is an rvalue).
`(a + 2)++;` `++(a + 2);`	Postfix and prefix operators require lvalues; (a + 2) is an rvalue.

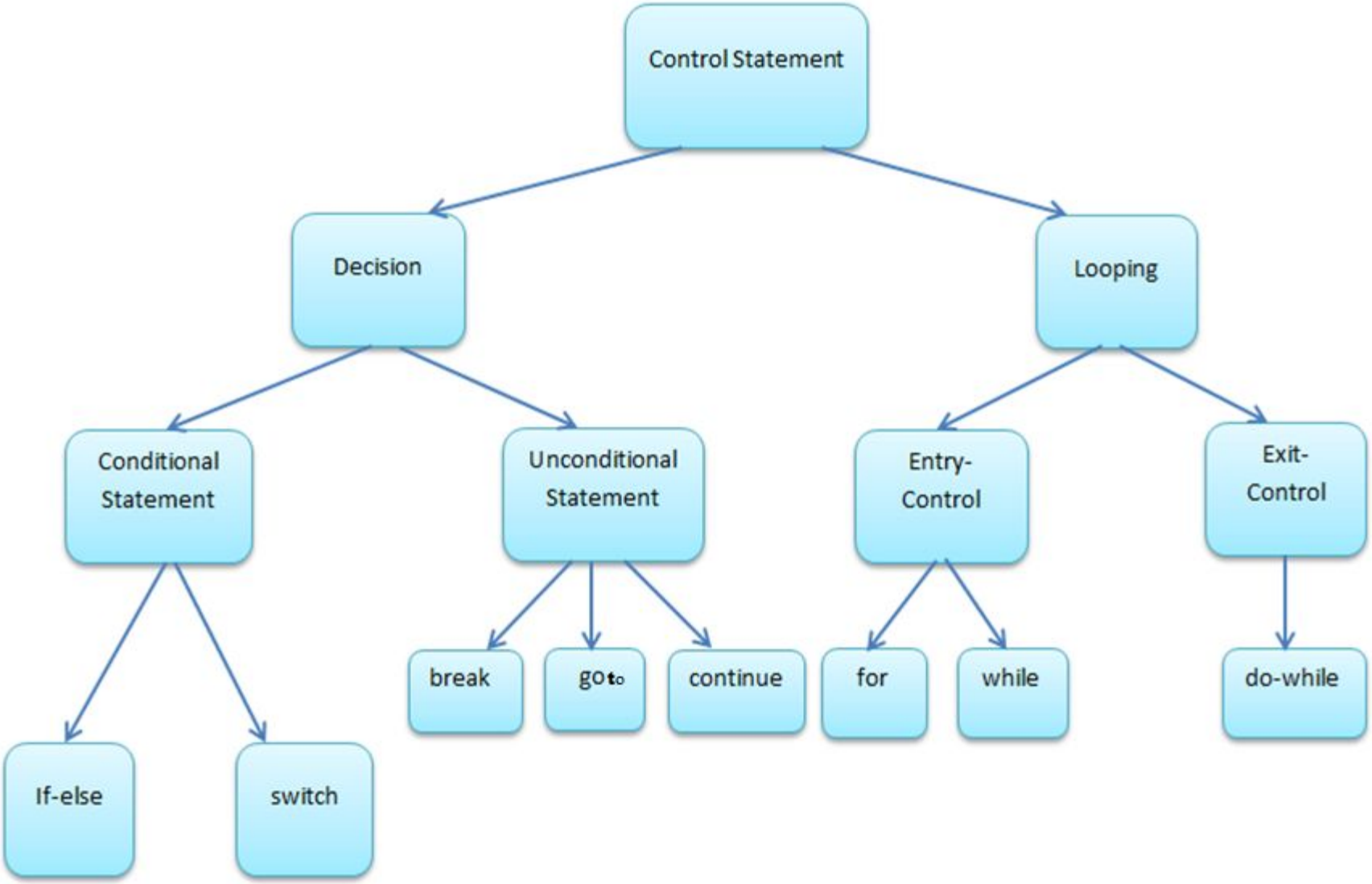
Table Invalid *rvalue* Expressions

SLO-1 :
Control Statements : if and else

Control Statements in C

- **Control Statements** in C are used to execute/transfer the control from one part of the program to another depending on a condition. These statements are also called as conditional statements.

Control Statements in C



Types of control statements

There are two types of Control Statements in C:

1. **if-else** statement
2. **switch** statement

if statement

So, the flow is that first the condition of if is checked and if it is true then statement(s) inside if are executed.

```
if ( condition )
   {
      statement;
      statement;
      statement;
      ....
   }
```

Example (if_1.c)

- Let's consider an example.

```
#include <stdio.h>
int main()
{
   int a = 5;
   int b = 5;
   clrscr();
   if ( a == b )
   {
      printf ( "a and b are equal\n" );
   }
  getch();
   return 0;
}
```

if (a==b) → a==b is the condition. Conditions are written inside '()'. Here, condition is true (will return 1). Since the condition is true, statements inside if will be executed. {} after if represents the body of if. Whatever is written inside '{}' is part of if.

if-else statement

- **if-else** statement is used to execute a statement block or a single statement depending on the value of a condition.
- **It has the following syntax**

```
if(condition)
   {
      statement;
      statement;
      ....
   }
   else
   {
      statement;
      statement;
      ....
   }
```

// where <condition> is a logical expression which will have the value true (1)or false (0)

Example (if_else_1.c)

```
#include <stdio.h>
int main()
{
   int a = 5;
   int b = 8;
  clrscr();
   if (a == b)
   {
      printf ("a and b are equal\n");
   }
   else
   {
      printf ("a and b are not equal\n");
   }
   getch();
   return 0;
}
```

The condition in *if* will be checked and since, it is false (5 and 8 are not equal), so statements in else are executed. It is that simple.

let's consider an example that a person is eligible to vote in an election only if his age is more than 18 years. (if_else_2.c)

```
#include <stdio.h>
int main()
{
   int age ;
   clrscr();
   printf ( " Enter your age " );
   scanf ( "%d" ,&age) ;
   if ( age >= 18 )
   {
      printf ( " Your age is 18+.\n" );
      printf ( " Eligible to vote\n" );
   }
   else
   {
      printf ( " Your age is not yet 18.\n" );
      printf ( " not eligible to vote\n" );
   }
   getch();
   return 0;
}
```

The condition in if is age >= 18. This means that if 'age' is more than or equal to 18 then it will print statements inside if, otherwise statements inside else will be printed.

SLO -2 :
else if and nested if, switch case

else if statements

There can be any number of else if between if and else.

```
if(condition)
   {
      statement
      statement
      ...
   }
   else if(condition)
   {
      statement
      statement
      ...
   }
   else if(condition)
   {
      statement
      statement
      ...
   }
   else
   {
      statement
      statement
      ....
   }
```

else if statements (if_else_3.c)

- If there is a single statement inside if, else or else if, we can skip the braces ({}). Let's look at an example.

```
#include <stdio.h>
int main()
{
   int a;
   clrscr();
   printf(“Enter the value\n”);
   scanf(“%d”,&a);
   if(a==30)
   {
      printf("It is 30\n");
   }
   else if(a==10)
   {
      printf("It is 10\n");
   }
   else if(a==5)
   {
      printf("It is 5\n");
   }
  getch();
   return 0;
}
```

As you can see in the above example that a single statement is considered as a part of if, else and else if respectively without any braces ({}).

Nested if/else (nestedif.C)

- We can also use if or else inside another if or else. See the example to print whether a number is greatest or not to understand it.

```
#include <stdio.h>
int main()
{
   int a = 8 ;
   int b = 4 ;
   int c = 10 ;
   clrscr();
   if ( a > b )
   {
      if ( a > c )
      {
          printf ( "a is the greatest number.\n" ) ;
      }
   }
  getch();
   return 0;
}
```

Nested if/else (nestedif.C)

- In the above example, the first expression i.e., (a>b) is true. So the statements enclosed in curly brackets {} of the first if condition are executed.

- Within the curly brackets, the first statement i.e., if(a>c) will be executed first. Since this condition is false, the statement printf(" a is the greatest number."); within the curly brackets of this if condition will not be executed.

- So, first we checked if 'a' is greater than 'b' or not, if it is, then we compared it with 'c'.

Finding greatest number by taking input by the user

```
#include <stdio.h>                    //greatestno.C
int main()
{
   int a = 8 ;
   int b = 4 ;
   int c = 10 ;
   clrscr();
   printf("Enter three numbers\n");
   scanf("%d %d %d",&a,&b,&c);
   if(a>b && a>c)
   {
      printf("%d is greatest\n",a);
   }
   else if(b>a && b>c)
   {
      printf("%d is greatest\n",b);
   }
   else if(c>a && c>b)
   {
      printf("%d is greatest\n",c);
   }
  getch();
   return 0; }
```

Finding greatest number by taking input by the user

- In the above example, we have three numbers a, b and c and we have to find the greatest among them.
- For that, we will first compare the first number with other numbers i.e., 'a' with 'b' and 'c' both.
- Now, if the condition (a>b && a>c) is true (which means that a is the greatest), then the statements enclosed in the curly brackets {} of the first if condition will be executed.
- If not so, then it will come to else if and check for (b>a && b>c).
- If this condition is true, then corresponding statements will be executed otherwise it will check the condition given in the last else if.

switch - case

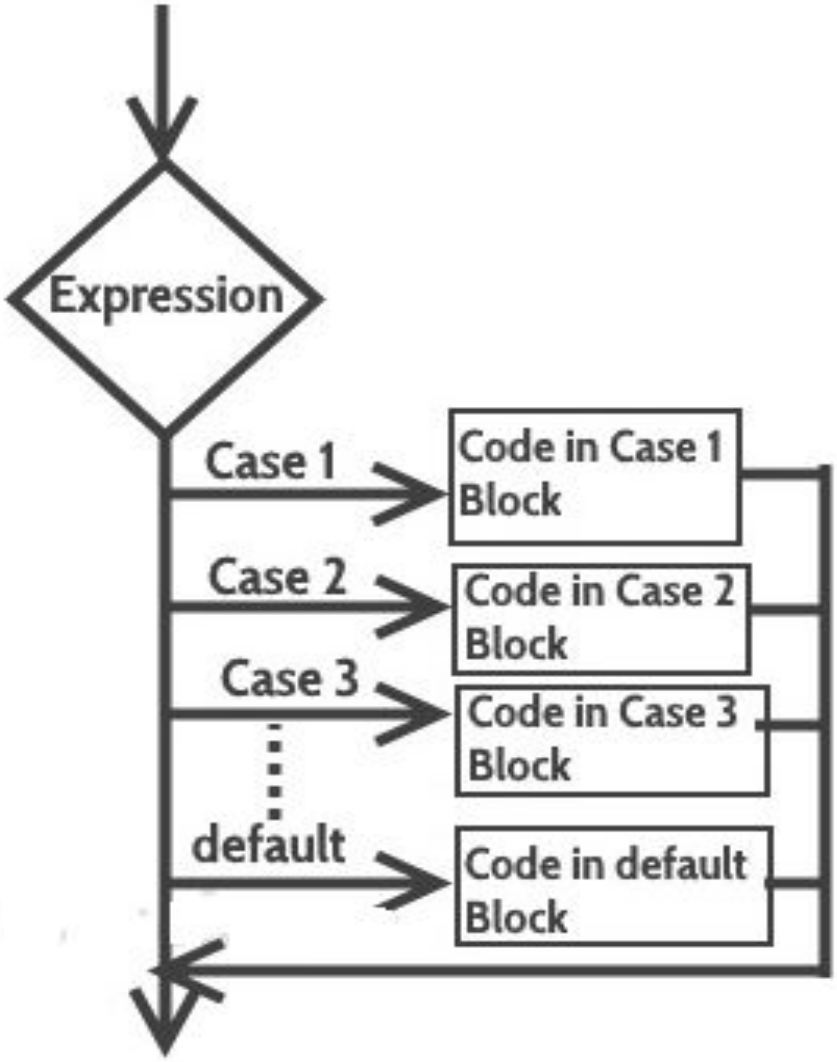
- Normally, if we have to choose one case among many choices, if-else is used. But if the number of choices is large, switch..case makes it a bit easier and less complex.
- switch...case is another way to control and decide the execution of statements other than if/else. This is used when we are given a number of choices (cases) and we want to perform a different task for each choice.

- Let's first have a look at its syntax.

Syntax : switch--case

```
switch(expression)
{
   case constant1:
      statement(s);
      break;
   case constant2:
      statement(s);
      break;
   /* you can give any number of cases */
   default:
      statement(s);
}
```

Flow Diagram of Switch Case



switch--case

- In switch...case, the value of expression enclosed within the brackets ( ) following switch is checked. If the value of the expression matches the value of constant in any of the case, the statement(s) corresponding to that case are executed.

- If expression does not match any of the constant values, then the statements corresponding to default are executed
- break is used to break or terminate a loop whenever we want and is also used with switch.

Example grades.c

```
#include <stdio.h>
int main()
{
   char grade ;
   printf("Enter your grade\n");
   scanf(" %c" , &grade);
   switch(grade)
   {
      case 'A':
         printf("Excellent!\n");
         break;
      case 'B':
        printf("Outstanding!\n");
        break;
      case 'C':
        printf("Good!\n");
        break;
      case 'D':
        printf("Can do better\n");
        break;
      case 'E':
        printf("Just passed\n");
        break;
      case 'F':
        printf("You failed\n");
        break;
      default:
        printf("Invalid grade\n");
   }
   return 0;
}
```

Quadratic equation

The quadratic formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- **determines whether the discriminant (b2−4ac) is less than, greater than or equal to 0.**
  - **When b2−4ac=0 there is one real root.**
  - **When b2−4ac>0 there are two real roots.**
  - **When b2−4ac<0 there are two complex roots.**

Find the Solution for $4x^2-2x+10=0$ using the Quadratic Formula where a = 4, b = -2, and c = 10

Find the Solution for

$$4x^2 - 2x + 10 = 0$$

using the Quadratic Formula where
a = 4, b = -2, and c = 10

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x = \frac{-(-2) \pm \sqrt{(-2)^2 - 4(4)(10)}}{2(4)}$$

$$x = \frac{2 \pm \sqrt{4 - 160}}{8}$$

$$x = \frac{2 \pm \sqrt{-156}}{8}$$

The discriminant $b^2 - 4ac < 0$
so, there are two complex roots.

$$x = \frac{2 \pm 2\sqrt{39}\,i}{8}$$

$$x = \frac{2}{8} \pm \frac{2\sqrt{39}\,i}{8}$$

Simplify fractions and/or signs:

$$x = \frac{1}{4} \pm \frac{\sqrt{39}\,i}{4}$$

which becomes

$$x = 0.25 + 1.56125\,i$$

$$x = 0.25 - 1.56125\,i$$

SLO-1 :
Iterations, Conditional and Unconditional branching

Iterative Statement

- The repeated execution of a statement or compound statement is accomplished by iteration zero, one, or more times.
- Iteration is the very essence of the power of computer.
- The repeated execution of a statement is often accomplished in a functional language by recursion rather then by iteration.
- General design issues for iteration control statements:

1. How is iteration controlled?
2. Where is the control mechanism in the loop?

Iterative Statement

- The primary possibilities for iteration control are logical, counting, or a combination of the two.
- The main choices for the location of the control mechanism are the top of the loop or the bottom of the loop.
- The body of a loop is the collection of statements whose execution is controlled by the iteration statement
- The term pretest means that the loop completion occurs before the loop body is executed.
- The term posttest means that the loop completion occurs after the loop body is executed.
- The iteration statement and the associated loop body together form an iteration construct.

Conditional and Unconditional branching

Branching statements are classified into two types

1. Conditional branching
2. Unconditional branching

1. Conditional Branching:

It includes the statements which works on checking the condition of a given expression to execute the code.

They are:

1. If
2. If-else
3. Nested if
4. switch
5. else-if ladder

Conditional and Unconditional branching

2. Unconditional Branching:

These statements doesn't check any condition to execute the code. These statements are pretty much useful to transfer the control from one block to another block over the program.

They are:

1. goto
2. return
3. continue
4. break

SLO-2 :
for loop

Looping Statements in C Language

- **Loop Statements** in C are used to execute and repeat a block of statements depending on the value of a condition.
- There are three types of Loop Control Statements in C language:

1. **for** loop
2. **while** loop
3. **do-while** loop

for loop

- The **for** loop is entry-controlled loop that provides a more concise(summarizing) loop control structure.
- A **for** loop is used to execute and repeat a block of statements depending on a condition. It has the following syntax:

```
for(<initial value>;<condition>;<increment/decrement>)
{
- - - - - - - - - -
<statement block>
- - - - - - - - - -
}
```

for loop

```
for(initialization; condition;  increment /propagation)
{
   statement(s)
}
```

- As stated, for is also a loop which repeats the codes inside its body.
- The code is repeated until the condition is true.
- Anything written in the place of initialization is executed only once and before starting the loop.
- The statement written at the place of propagation is executed after every iteration of the loop.

Example : for loop (forloop1.c)

```
#include <stdio.h>
int main()
{
   int a ;
   clrscr();
   for ( a = 1 ; a <= 10 ; a ++ )
   {
       printf ( "Hello World\n" ) ;
   }
   getch();
   return 0;
}
```

Example : for loop (forloop1.c)

- Now let's see how for loop works.

  for(a=1; a<=10; a++)

- a=1 → This is the initialization of the loop and is executed once at the starting of the loop. Generally, it used to assign value to a variable. Here, 'a' is assigned a value 1.
- a<=10 → This is the condition which is evaluated. If the condition is true, the statements written in the body of the loop are executed.
- If it is false, the statement just after the for loop is executed. This is similar to the condition we used in the while loop which was being checked again and again.

Example : for loop (forloop1.c)

- a++ → This is executed after the execution of the code in the body of for loop. In this example, the value of 'a' increases by 1 every time the code in the body of for loop executes. There can be any expression here which you want to run after every iteration of the loop.

- In the above example, firstly a=1 assigns the value of 1 to the variable a.

- Then condition a<=10 is checked. Since the value of 'a' is 1, therefore the code in the body of for loop is executed and 'Hello World' gets printed.

Example : for loop (forloop1.c)

- Once the codes in the body of for loop are executed, a++ is executed which increases the value of 'a' by 1. Now, the value of 'a' is 2.

- Again the condition a<=10 is checked which is true because the value of 'a' is 2. Again codes in the body of for loop are executed and 'Hello World' gets printed and then the value of 'a' is again increased by 1 (a++).

- When the value of 'a' becomes 10, the condition a <= 10 evaluates to true and 'Hello World' gets printed. Now, when a++ increases the value of 'a' to 11, the condition a<=10 becomes false and the loop terminates.

Example : for loop (forloop2.C)

```
#include <stdio.h>
int main()
{
   int a ;
   clrscr();
   printf("Table of 12\n");
   for ( a = 1 ; a <= 10 ; a ++ )
   {
       printf ( "12*%d = %d\n",a,12*a ) ;
   }
   getch();
   return 0;
}
```

Nested for loop

- This statement block of a for loop lies completely inside the block of another for loop that is called **Nested for** loop or **Nested for** Statement.

```
for(i=1;i<=3;i++) // outer loop
{
- - - - - - - -
for(j=1;j<=5;j++) // inner loop
{
- - - - - - - -
<statement block>
- - - - - - - -
}
}
```

Example : Nested for loop

```
                                        //nestedloop4.C
#include <stdio.h>
int main()
{
   int i;
   int j;
   clrscr();
   for(i = 12;i<=14;i++)
      {
            /*outer loop*/
            printf("Table of %d\n",i);
                  for(j = 1;j<=10;j++)
                     {
                        /*inner loop*/
                              printf("%d*%d\t=\t%d\n",i,j,i*j);
                  }
Getch();
        }
  getch();
   return 0;
}
```

Example : Nested for loop

- At the initial execution of the first, 'i' is 1 and thus, 'Table of 12' gets printed.
- Now, the next statement is for(j = 1;j<=10;j++). So, executing this statement:
- 'j' is 1 and 12*1 = 12 will be printed.
- Now, the second iteration of the inner loop will take place. 'i' is still 12 but 'j' will be increased to 2. So, 12*2 = 24 will be printed.
- Coming to the last iteration of the inner loop, 'i' is still 12, and 'j' will be 10, so 12*10 = 120 will be printed.
- Now, coming out of the outer loop, there is no statement after the inner loop, so 'i' will be increased to 13 (i++) for the next iteration of the outer loop and 'Table of 13' will be printed and things will be repeated with the value of 'j' equal to 13.

Example : Nested for loop

- Suppose we have to change the values of a variable 'a' from 1 to 5 and the values of another variable 'b' from 5 to 1.
- //nested_3.C

```
#include <stdio.h>
int main()
{
   int a ;
   int b ;
   for ( a = 1 , b = 5 ; a <= 5 , b >= 1 ; a ++ , b -- )
   {
       printf ( "a = %d\t b = %d\n" , a , b ) ;
   }
   return 0;
}
```

SLO-1 :
while loop

while loop

- It is also an entry-controlled loop statement.
- A **while** loop is used to execute and repeat a block of statements depending on a condition. It has the following syntax:

```
while(condition)
{
   statement(s)
}
```

while loop…

- while loop checks whether the condition written in '( )' is true or not.
- If the condition is found true, then statements written in the body of the while loop i.e., inside the braces { } are executed.
- Then, again the condition is checked, and if found true then statements in the body of the while loop are executed again.
- This process continues until the condition becomes false.

Example : while loop... (whileexample.c)

```
#include <stdio.h>
int main()
{
   int a = 1;
    clrscr();
   while ( a <= 10 )
   {
      printf ( "Hello World\n" );
      a ++;
   }
  getch();
   return 0;
}
```

Example : while loop...

- For better understanding, let's understand our example step by step.
- In our example, firstly, we assigned a value of 1 to a variable 'a'.
- while(a <= 10) checks the condition 'a <= 10'. Since the value of a is 1 which is less than 10, the statements of while (within the braces { }) are executed.
- 'Hello World' is printed and a++ increases the value of 'a' by 1. So, now the value of 'a' becomes 2.
- Now, again the condition is checked. This time also a <= 10 is true because the value of 'a' is 2. So, again 'Hello World' is printed and the value of 'a' increased to 3.
- When the value of 'a' becomes 10, again the condition a <= 10 is true and 'Hello World' is printed for the tenth time. Now, a++ increases the value of 'a' to 11.
- This time, the condition a <= 10 becomes false and the while loop terminates.

Example : while loop... (whileexample1.C)

```
#include <stdio.h>
int main()
{
   char choice = 'y'; clrscr();
   while(choice == 'y')
   {
      int a;
      printf("Enter a number to check odd or even\n");
      scanf("%d",&a);
      if(a%2==0)
      {
         printf("Your number is even\n");
      }
      else
      {
         printf("Your number is odd\n");
      }

      printf("Want to check more y for yes n for no\n");
      scanf(" %c",&choice);

   } getch();
   return 0;
}
```

SLO-2 :
do..while, goto, break, continue

do-while

- A **do-while** loop is an exit-controlled loop statement. It is used to execute and repeat a block of statements depending on a condition. It has the following syntax :

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

do-while (do_while.c)

```
#include <stdio.h>
int main()
{
   int a = 1;
   clrscr();
   do
   {
      printf( "Hello World\n" );
      a ++;
   }
   while(a <= 10);
    getch();
   return 0;
}
```

do-while

- Let's understand this.
- At first, the codes inside the body of loop (i.e. within the braces { } following do) are executed without checking condition. This prints 'Hello World' and a++ increments the value of 'a' by 1. So, the value of 'a' becomes 2.
- Once the code inside the braces ({ }) is executed, condition 'a <= 10' is checked. Since the value of 'a' is 1, so the condition is satisfied.
- Again the code inside the body of loop is executed and the value of 'a' becomes 2.
- When the value of 'a' is 10 and 'Hello World' is printed for the tenth time, a++ increases the value of 'a' to 11. After this, the condition becomes false and the loop terminates.

Comparison Between for loop while loop and do-while loop is given below:

for loop	while loop	do-while loop
1. A for loop is used to execute and repeat a statement block depending on a condition which is evaluated at the beginning of the loop.	1. A while loop is used to execute and repeat a statement block depending on a condition which is evaluated at the beginning of the loop.	1. A do-while loop is used to execute and repeat a statement block depending on a condition which is evaluated at the end of the loop.
2. A variable value is initialized at the beginning of the loop.	2. A variable value is initialized at the beginning or before of the loop.	2. A variable value is initialized before the loop or assigned inside the loop.
3. The statement block will not be executed when the value of the condition is false.	3. The statement block will not be executed when the value of the condition is false.	3. The statement block will not be executed when the value of the condition is false, but the block is executed at least once irrespective of the value of the condition.
4. A statement to change the value of the condition or to increment the value of the variable is given at the beginning of the loop.	4. A statement to change the value of the condition or to increment the value of the variable is given inside of the loop.	4. A statement to change the value of the condition or to increment the value of the variable is given inside of the loop.
5. A for loop is commonly used by many programmers.	5. A while loop is widely used by many programmers.	5. A do-while loop is used in some cases where the condition need to be checked at the end of the loop.

Unconditional Control Transfer Statements in C

- **Unconditional Control Transfer Statements** is used to transfers the control to some other place in the program.
- In C Programming Language, There are four types of unconditional control transfer statements.

1. **goto**
2. **break**
3. **continue**
4. **return**

goto statement

The **goto** statement is used to transfer the control from one part of the program to another. It has the following syntax :

```
goto label_name;
..
..
label_name: C-statements
```

```
- - - - - - - - - - -
goto display;
- - - - - - - - - - -
- - - - - - - - - - -
display:
```

when this statement is executed, the control is transferred to the statement label **display** which is followed by a colon.

Example : goto statement (goto.c)

```
/*To print numbers from 1 to 10 using goto statement*/
#include <stdio.h>
int main()
{
   int number;
   clrscr();
   number=1;

repeat:
   printf("%d\n",number);
   number++;

   if(number<=10)
      goto repeat;
    getch();
   return 0;
}
```

break statement

- The **break** statement is used to transfer the control to the end of a statement block in a loop.
- It is an unavoidable statement to transfer the control to the end of a switch statement after executing anyone statement block.

It has the following syntax

break;

Continue statement

- The **continue** statement is used to transfer the control to the beginning of a statement block in a loop.
- It has the following syntax:
  continue;

Example : Continue statement (continue.c)

```
/**
 * C program to print even numbers between 1 to 100
 */
#include <stdio.h>

int main()
{

   int num;
   clrscr();
   printf("Even numbers between 1 to 100: \n");

   for(num=1; num<=100; num++)
   {

      if(num % 2 == 1)
         continue;

      printf("%d ", num);
   }
   getch();
```

return statement

- The **return** statement terminates the execution of a function, it returns control to the calling function.
- It resumes the execution in the calling function immediately. It has the following syntax:

jump instruction **or** statement:

return expression;

SLO-1 :
Array Basic and Types

Arrays

- An array is a collection of data elements that are of the same type (e.g., a collection of integers, collection of characters, collection of doubles).

Currency	U.S. $	Aust $	U.K. £	Can $	DMark	FFranc	¥en	SFranc	Euro
Last Trade	N/A	Oct 14	Oct 14	Oct 14	Oct 14	Oct 14	Oct 14	Oct 14	12:39AM
U.S. $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082
Aust $	1.54	1	2.562	1.04	0.8491	0.2532	0.01435	1.045	1.666
U.K. £	0.6012	0.3904	1	0.4058	0.3314	0.09883	0.005601	0.4079	0.6505
Can $	1.481	0.9619	2.464	1	0.8167	0.2435	0.0138	1.005	1.603
DMark	1.814	1.178	3.017	1.224	1	0.2982	0.0169	1.231	1.963
FFranc	6.083	3.95	10.12	4.106	3.354	1	0.05667	4.127	6.582
¥en	107.3	69.7	178.5	72.46	59.18	17.64	1	72.82	116.1
SFranc	1.474	0.9571	2.452	0.995	0.8126	0.2423	0.01373	1	1.595
Euro	0.9242	0.6001	1.537	0.6238	0.5095	0.1519	0.00861	0.627	1

Arrays

- Array is the collection of similar data types or collection of similar entity stored in contiguous memory location.
- Array of character is a string. Each data item of an array is called an element.
- And each element is unique and located in separated memory location.
- Each of elements of an array share a variable but each element having different index no. known as subscript.
- An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension. And number of subscript is always starts with zero.
- One dimensional array is known as vector and two dimensional arrays are known as matrix.

Arrays

- 1-dimensional array.

Currency Last Trade	U.S. $ N/A	Aust $ Oct 14	U.K. £ Oct 14	Can $ Oct 14	DMark Oct 14	FFranc Oct 14	¥en Oct 14	SFranc Oct 14	Euro 12:39AM
U.S. $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082

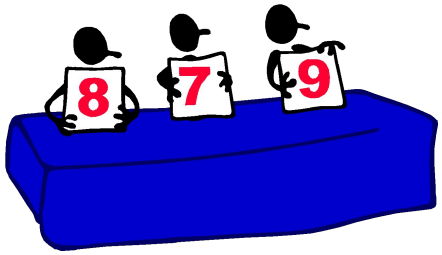
- 3-dimensional array (3rd dimension is the day).

Currency Last Trade	U.S. $ N/A	Aust $ Oct 14	U.K. £ Oct 14	Can $ Oct 14	DMark Oct 14	FFranc Oct 14	¥en Oct 14	SFranc Oct 14	Euro 12:39AM
U.S. $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082
Aust $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082
U.K. £	0.	0.6	1.	1.	6.	10	1.	0.9	
Can $	1	0.6493	1.663	0.675	0.5513	0.1644	0.009316	0.6784	1.082
DMark	1.54	1	2.562	1.04	0.8491	0.2532	0.01435	1.045	1.666
FFranc	0.6012	0.3904	1	0.4058	0.3314	0.09883	0.005601	0.4079	0.6505
¥en	1.481	0.9619	2.464	1	0.8167	0.2435	0.0138	1.005	1.603
SFranc	1.814	1.178	3.017	1.224	1	0.2982	0.0169	1.231	1.963
Euro	6.083	3.95	10.12	4.106	3.354	1	0.05667	4.127	6.582
	107.3	69.7	178.5	72.46	59.18	17.64	1	72.82	116.1
	1.474	0.9571	2.452	0.995	0.8126	0.2423	0.01373	1	1.595
	0.9242	0.6001	1.537	0.6238	0.5095	0.1519	0.00861	0.627	1

Oct 14

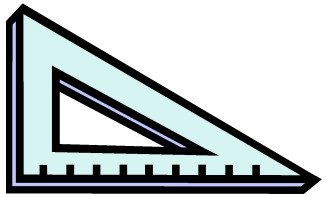
Oct 15

Oct 16



Array Applications

- Given a list of test scores, determine the maximum and minimum scores.
- Read in a list of student names and rearrange them in alphabetical order (sorting).
- Given the height measurements of students in a class, output the names of those students who are taller than average.
- **ADVANTAGES:** array variable can store more than one value at a time where other variable can store one value at a time.



Array Types

Arrays can of following types:

1. One dimensional (1-D) arrays or Linear arrays
2. Multi dimensional arrays
   (a) Two dimensional (2-D) arrays or Matrix arrays
   (b) Three dimensional arrays

One dimensional (1-D) arrays or Linear arrays:

- In it each element is represented by a single subscript. The elements are stored in consecutive memory locations.

  E.g. A [1], A [2], ….., A [N].

Multi dimensional arrays

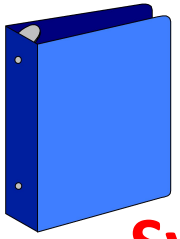
(a) Two dimensional (2-D) arrays or Matrix arrays:

- In it each element is represented by two subscripts. Thus a two dimensional m x n array A has m rows and n columns and contains m*n elements.
- It is also called matrix array because in it the elements form a matrix. E.g. A [2] [3] has 2 rows and 3 columns and 2*3 = 6 elements.

(b) Three dimensional arrays:

- In it each element is represented by three subscripts. Thus a three dimensional m x n x l array A contains m*n*l elements. E.g. A [2] [3] [2] has 2*3*2 = 12 elements.

SLO-2 :
Array Initialization and Declaration



Array Declaration

Syntax:

<type> <arrayName>[<array_size>]

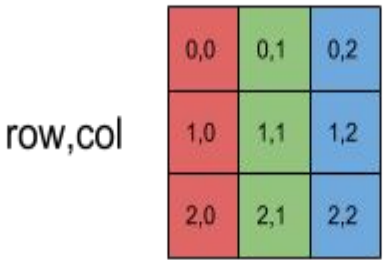
Ex. int Ar[10];

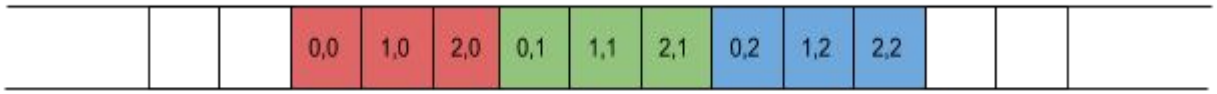
- The array elements are all values of the type **<type>.**
- The size of the array is indicated by **<array_size>**, the number of elements in the array.
- **<array_size>** must be an **int** constant or a constant expression. Note that an array can have multiple dimensions.

Array Declaration …

- Generic form:–
  - *ArrayName*[*integer-expression*]
  - *ArrayName*[*integer-expression*] [*integer-expression*]

  – Same type as the underlying type of the array
- Definition:– *Array Index* – the expression between the square brackets





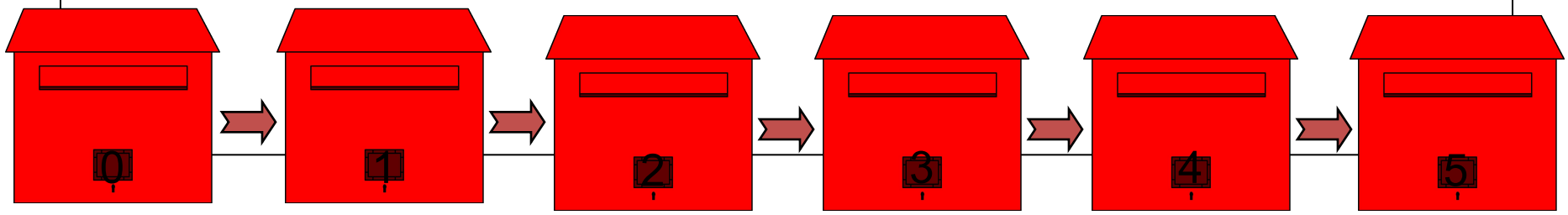
Array Declaration

- The declaration of an array tells the compiler that, the data type, name of the array, size of the array and for each element it occupies memory space.
- Like for int data type, it occupies 2 bytes for each element and for float it occupies 4 byte for each element etc.
- The size of the array operates the number of elements that can be stored in an array and it may be a int constant or constant int expression

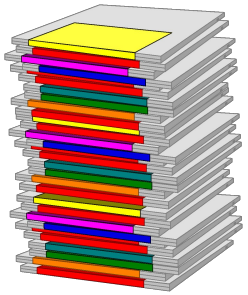
Array Declaration

```
// array of 10 uninitialized ints
int Ar[10];
```

	0	1	2	3	4	5	6	7	8	9
Ar	--	--	--	--	--	--	--	--	--	--



Subscripting



- Declare an array of 10 integers:

  int Ar[10]; // array of 10 ints

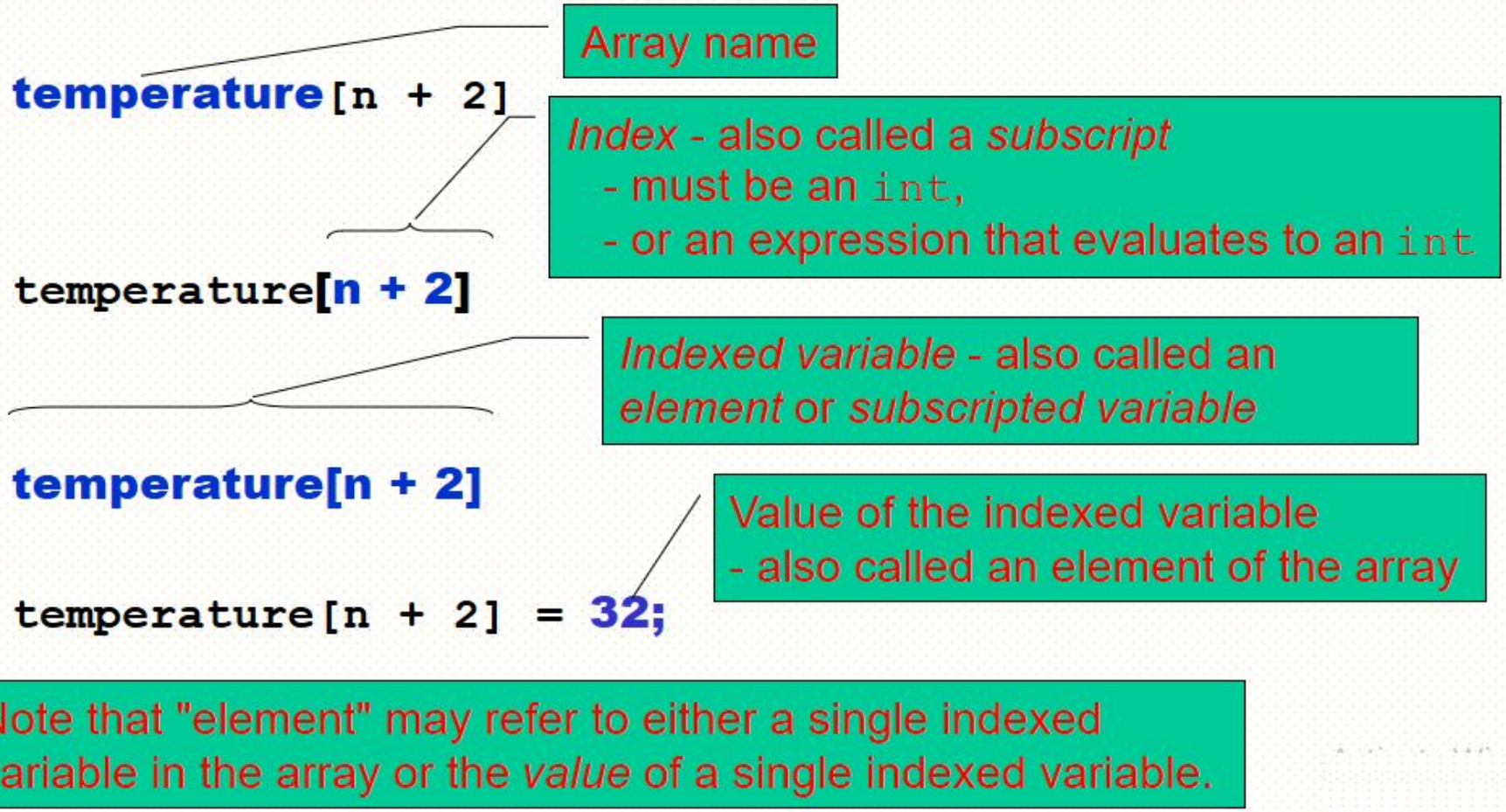
- To access an individual element we must apply a **subscript** to array named **Ar.**
  - A subscript is a bracketed expression.
    - The expression in the brackets is known as the **index.**
  - First element of array has **index 0**.

    Ar[0]
  - Second element of array has **index 1, and so on**.

    Ar[1], Ar[2], Ar[3],…
  - Last element has an index one less than the size of the array.

    Ar[9]
- Incorrect indexing is a common error.

Some Array Terminology

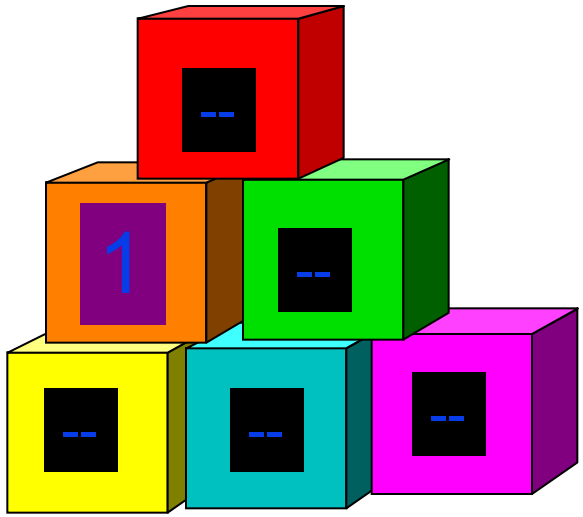


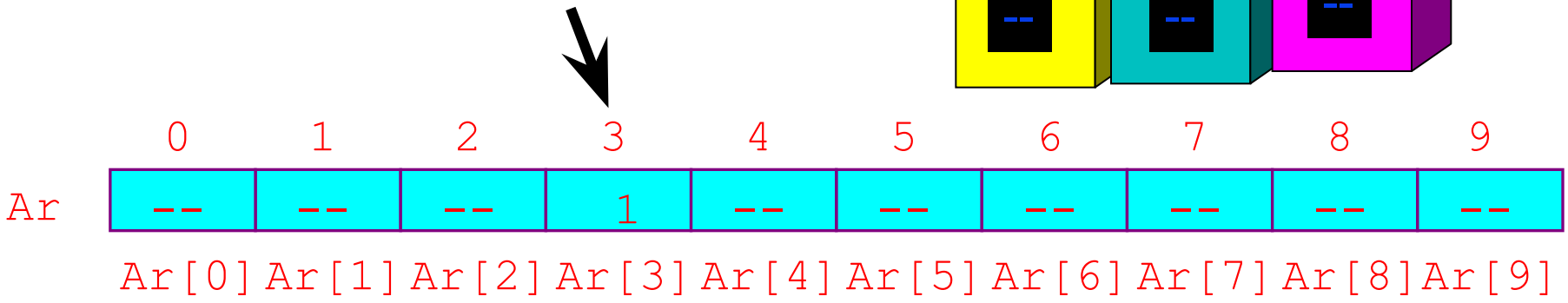
Note that "element" may refer to either a single indexed variable in the array or the *value* of a single indexed variable.

Subscripting

```
// array of 10 uninitialized ints
int Ar[10];

Ar[3] = 1;
int x = Ar[3];
```



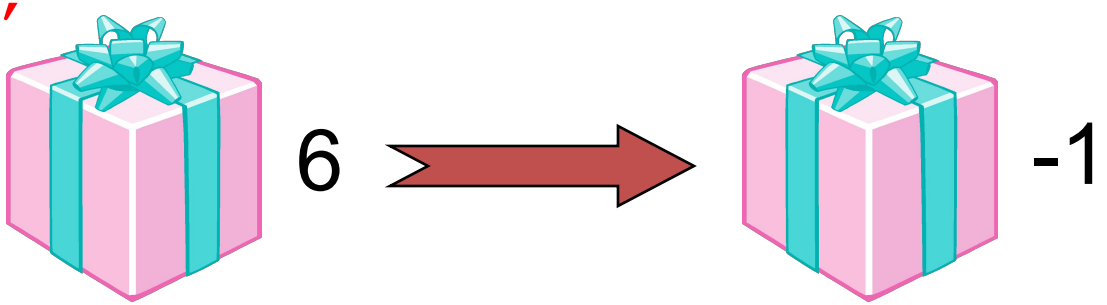


Array Initialization

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

	0	1	2	3	4	5	6	7	8	9
Ar	9	8	7	6	5	4	3	2	1	0

```
Ar[3] = -1;
```



	0	1	2	3	4	5	6	7	8	9
Ar	9	8	7	-1	5	4	3	2	1	0

Total size in byte for 1D array is:

- Total bytes=size of (data type) * size of array. Example : if an array declared is:
- int [20]; Total byte= 2 * 20 =40 byte.

Examples

int A[10]

- An array of ten integers
- **A[0], A[1], …, A[9]**

double B[20]

- An array of twenty long floating point numbers
- **B[0], B[1], …, B[19]**

- Arrays of **structs, unions, pointers,** etc., are also allowed
- Array indexes *always* start at zero in *C*

Examples (continued)

int C[]

- An array of an unknown number of integers (allowable in a parameter of a function)
- **C[0]**, **C[1]**, …, **C[***max-1***]**

int D[10][20]

- An array of ten rows, each of which is an array of twenty integers
- **D[0][0]**, **D[0][1]**, …, **D[1][0]**, **D[1][1]**, …, **D[9][19]**
- Not used so often as arrays of pointers

Array Element

- May be used wherever a variable of the same type may be used
  - In an expression (including arguments)
  - On left side of assignment

- **Examples:–**

```
A[3] = x + y;
x = y – A[3];
z = sin(A[i]) + cos(B[j]);
```

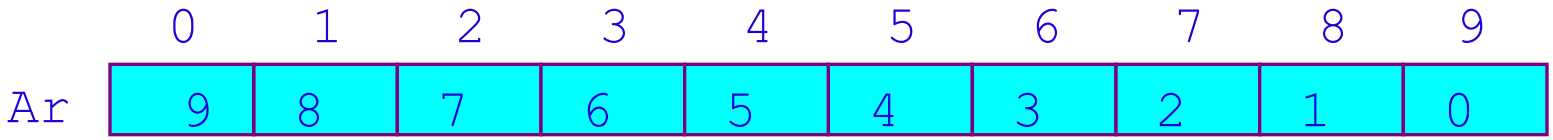
SLO-1 :
Initialization : one Dimensional Array

Arrays in C programming

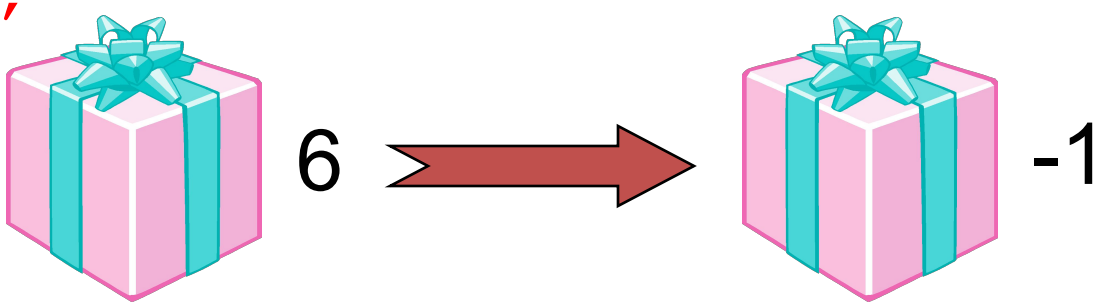
- An array is a group (or collection) of same data types. For example an int array holds the elements of int types while a float array holds the elements of float types.

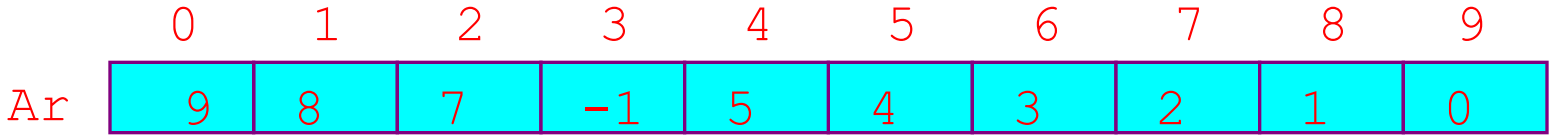
Initialization : 1-D Array

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




```
Ar[3] = -1;
```





SLO-2 :
Accessing, Indexing one Dimensional Array operations

How to declare Array in C

- int num[35]; /* An integer array of 35 elements */
- char ch[10]; /* An array of characters for 10 elements */

Similarly an array can be of any data type such as double, float, short etc.

How to access element of an array in C

- You can use array subscript (or index) to access any element stored in array. Subscript starts with 0, which means arr[0] represents the first element in the array arr.

  In general arr[n-1] can be used to access nth element of an array. where n is any integer number.

For example:

```
int mydata[20];
mydata[0] /* first element of array mydata*/
mydata[19] /* last (20th) element of array
mydata*/
```

Example of Array In C programming to find out the average of 4 integers

```
#include <stdio.h>
int main()
{
   int avg = 0;
   int sum =0;
   int x=0;

   /* Array- declaration – length 4*/
   int num[4];

   /* We are using a for loop to traverse through the array
    * while storing the entered values in the array
    */
   for (x=0; x<4;x++)
   {
      printf("Enter number %d \n", (x+1));
      scanf("%d", &num[x]);
   }
   for (x=0; x<4;x++)
   {
      sum = sum+num[x];
   }
```

Example of Array In C programming to find out the average of 4 integers...

```
 avg = sum/4;
    printf("Average of entered number is: %d",
avg);
    return 0;
}
```

SLO-1 :
One Dimensional Array operations
SLO-2 :
Array Programs : 1D

Array operations

- Traversal
- Copying
- Reversing
- Sorting
- Insertion
- Deletion
- Searching
- Merging

Traversal:

- Traversal means accessing each array element for a specific purpose, either to perform an operation on them , counting the total number of elements or else using those values to calculate some other result.
- Since array elements is a linear data structure meaning that all elements are placed in consecutive blocks of memory it is easy to traverse them.

Algorithm:

Consider A[]is the array:

Step 1: Initialize counter c = lower_bound_index

Step 2: Repeat steps 3 to 4 while c < upper_bound

Step 3: Apply the specified operation on A[c]

Step 4: Increment counter : C = C + 1

[Loop Ends]

Step 5: Exit

Example: Write a program to calculate the average marks of a particular student: (array4.c)

```
#include<stdio.h>
#include<conio.h>
int main(){
int i, marks[5], n, sum = 0;
float avg;
printf("Enter the no.of subjects:\n");
scanf("%d",&n);
printf("Enter the marks obtained in your %d subjects\n", n);
for(i=0;i<n;i++)
{
scanf("%d", &marks[i]);
}
for(i=0;i<n;i++)
{
sum = sum + marks[i];
}
avg = (sum / n);
printf("Average of marks is : %.2f \n", avg);
return 0;
}
```

Input:

```
Enter the no.of subjects:
5
Enter the marks obtained in your 5 subjects
```

Copying elements of an array:

- Copying array elements to another array will yield an array of the same length and elements as the original one.
- The destination array should also be of the same or greater size as that of original array in order to hold the array contents.
- The copying of elements would be done on index by index basis.

Algorithm:

Step 1 : Take two arrays A, B

Step 2 : Store values in A

Step 3 : Loop for each value of A

Step 4 : Copy each index value to B array at the same index location

Example-Copying elements (array5.c)

```
#include<stdio.h>
int main()
{
int arr1[20], arr2[20], i, num;
printf("\nEnter the no of elements in the array :");
scanf("%d", &num);
//Accepting values into Array
printf("\nEnter the array elements :");
for (i = 0; i < num; i++) {
scanf("%d", &arr1[i]);
}
// Copying data from source array A to destination array 'b
for (i = 0; i < num; i++) {
arr2[i] = arr1[i];
}
//Printing of all elements of array
printf("The copied array is as follows:");
for (i = 0; i < num; i++)
printf("\narr2[%d] = %d", i, arr2[i]);
return (0);
}
```

Input:

```
Enter the no of elements in the array :5
Enter the array elements :
1
2
3
4
5
```

Reversing elements of an array:

- Reversing an array means that the sequence of elements of array will be reversed.
- For instance if your array ‘A’ has two elements : A[0] = 1; A[1] = 2; then after reversal A[0] = 2 and A[1] = 1.

Algorithm

- Take input from user into array A.
- Store value of element of A in B, starting with last element of A and placing it as first element in B.
- Loop for each value of A.
- Store each value of element B into A as it is. (Copying the reversed array back to source array).
- Loop for each value of B.

Example-Reversing (array6.c)

```
#include <stdio.h>
int main()
{
int n, i, j, a[20], b[20];
printf("Enter the number of elements in array\n");
scanf("%d", &n);
printf("Enter array elements\n");
for (i = 0; i < n ; i++)
scanf("%d", &a[i]);
//Copying elements into array b starting from end of array a
for (i = n - 1, j = 0; i >= 0; i--, j++)
b[j] = a[i];
//Copying reversed
for (i = 0; i < n; i++)
a[i] = b[i];
printf("Reversed array is\n");
for (i = 0; i < n; i++)
printf("%d\n", a[i]);
return 0;
}
```

Input:

```
Enter the number of elements in array
5
Enter array elements
11
12
13
14
```

Sorting elements of an array:

- Sorting elements if array means to order the elements in ascending or descending order – usually in ascending order.
- The basic approach to sorting is Bubble sort method where in nested loop is used to sort elements of array.

Algorithm

- Create an array of fixed size.
- Take n, a variable which stores the number of elements of the array, less than maximum capacity of array.
- Iterate via for loop to take array elements as input, and print them.
- The array elements are in unsorted fashion, to sort them, make a nested loop.
- In the nested loop, the each element will be compared to all the elements below it.
- In case the element is greater than the element present below it, then they are interchanged.
- After executing the nested loop, we will obtain an array in ascending order arranged elements.

Example-Sorting (array7.c)

```
#include <stdio.h>
int main()
{
int i, j, temp, n, arr[30];
printf("Enter the number of elements in your array: \n");
scanf("%d", &n);
printf("Enter the array elements: \n");
for (i = 0; i < n; ++i)
scanf("%d", &arr[i]);
for (i = 0; i < n; ++i)
{
for (j = i + 1; j < n; ++j)
{
if (arr[i] > arr[j])   //to check if current element greater than i+1th element, if yes; perform swap.
{ temp =  arr[i];
arr[i] = arr[j];
arr[j] = temp;
}}}
printf("\nThe array sorted in ascending order is as given below: \n");

for (i = 0; i < n; ++i)
printf("%d\n", arr[i]);
return 0;
}
```

Input:

```
Enter the number of elements in your array
5
Enter the array elements
22
10
73
```