Type Casting

Type Casting

- Type casting is a way to convert a variable from one data type to another data type.
- For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.
- It is of two types:
    - Implicit Type Cast
    - Explicit Type Cast
- Implicit type casting is done by the compiler and there is no data loss.
- Explicit type casting is done by the programmer and there may be some data loss.
- We can convert the values from one type to another explicitly using the **cast operator**

Type Casting

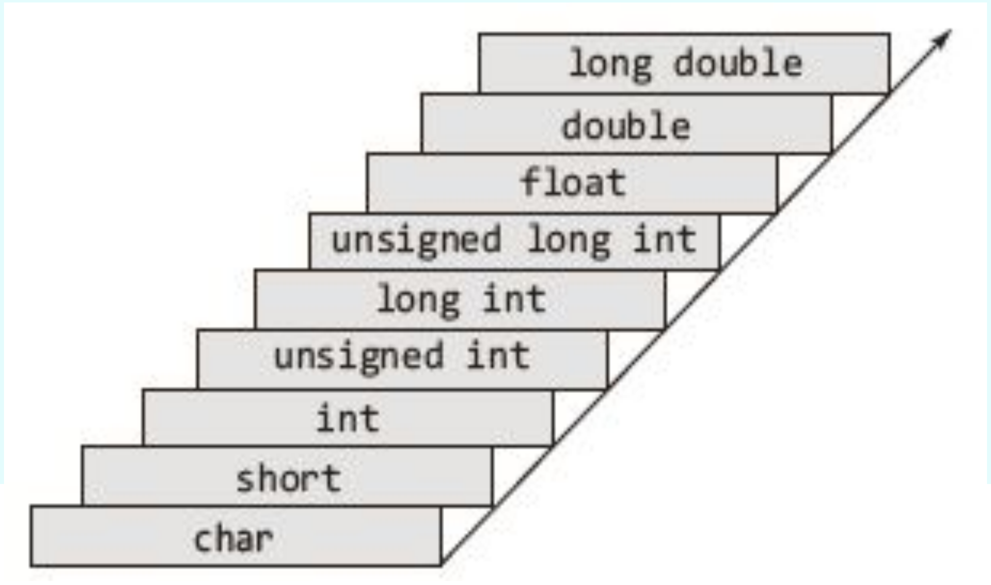
```
#include <stdio.h>
int main()
{
int sum = 17, count = 5;
double avg;
avg = (double) sum / count;
printf("Value of avg : %f\n", avg);
return 0;
 }
```

Output

```
Value of avg : 3.400000
```

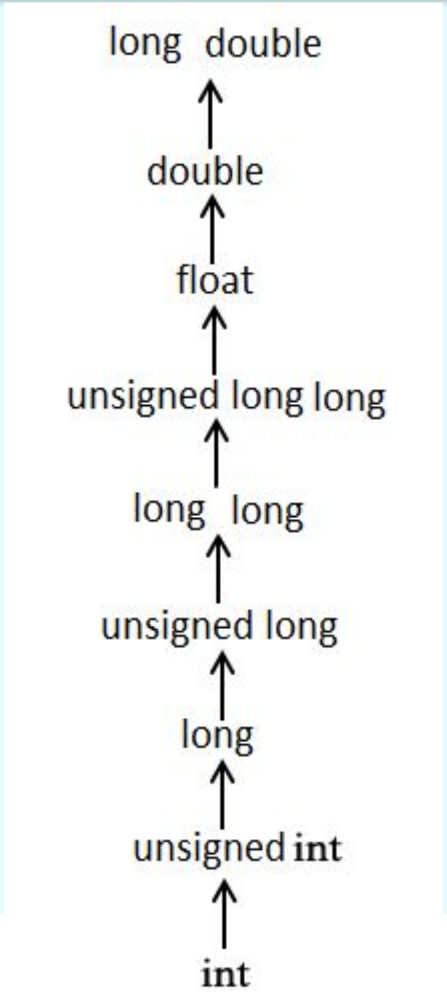
Type Conversion

- When a C expression is evaluated, the resulting value has a particular data type.
- If all the variables in the expression are of the same type, the resulting type is of the same type as well.
- For example, if x and y are both of int type , the expression x +y is of int type as well.
- The smallest to the largest data types conversion with respect to size is along the arrow as shown below:



RULE : Type Conversion

- The **usual arithmetic conversions** are implicitly performed to cast their values to a common type. The compiler first performs *integer promotion (convert to int)*; if the operands still have different types, then they are converted to the type that appears highest in the following hierarchy
- **char or short** (signed or unsigned) are converted to **int** (signed or unsigned).
- **float operands are converted to double.**
- **If any one operand is double, the other operand is also converted to double**, and that is the type of the result;
- **If any one operand is long, the other operand is treated as long**, and that is the type of the result;
- **If any one operand is of type unsigned, the other operand is converted to unsigned**, and that is also the type of the result.



RULE : Type Conversion

```
#include <stdio.h>
 main()
{
    int i = 17;
    char c = 'c'; /* ascii value is 99 */
    float sum;

    sum = i + c;
     printf("Value of sum : %f\n", sum );
}
```

When the above code is compiled and executed, it produces the following result –

```
Value of sum : 116.000000
```

Program Statements

- A statement is a syntactic constructions that performs an action when a program is executed.
- All C program statements are terminated with a semi-colon (;)

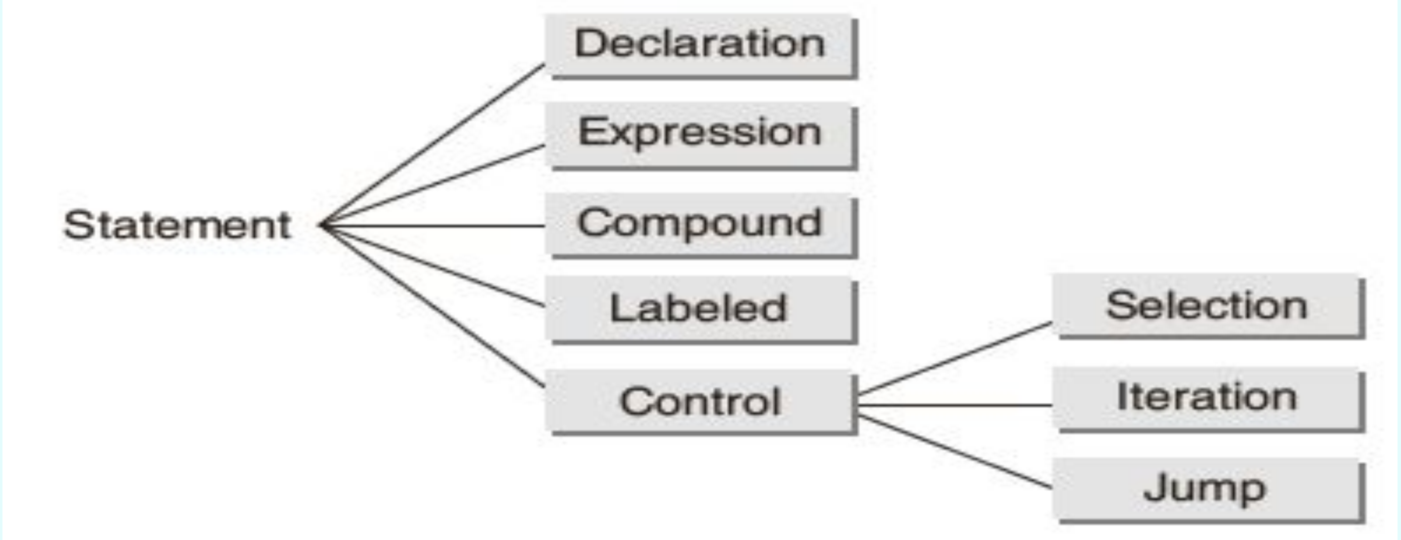


Figure Different types of program statements available in C

Program Statements

- ***Declaration :*** It communicates the information about the name and type of the data objects needed during program execution to the language translator.
  - int a;
  - int b;
  - int c;

  Or int a,b,c;

  This line informs the C compiler that it needs to allocate space for integers
- ***Expression statement:*** An *expression is a sequence of operators and* operands that specifies computation of a value . Example: x = 4

Program Statements

- ***Compound statement is a sequence of statements that*** may be treated as a single statement in the construction of larger statements.
- A compound statement (also called a "block") typically appears as the body of another statement, such as the **if** statement
- if (i > 0)

```
 {
    line[i] = x;
    x++;
    i--;
 }
```

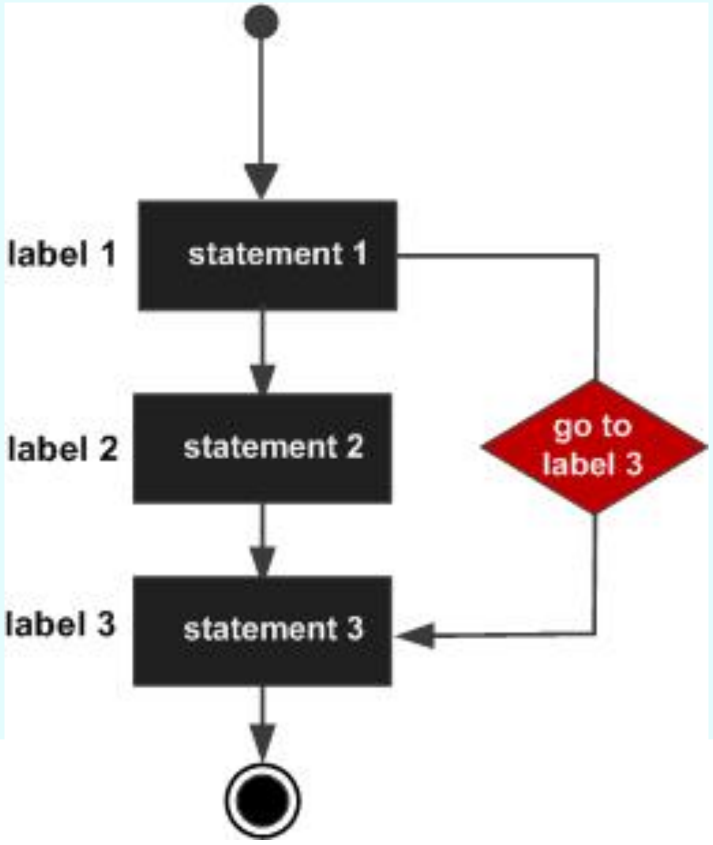
Program Statements

- ***Labelled statements can be used to mark any statement so*** that control may be transferred to the statement.

- goto label;

  .. .

  label: statement;



Program Statements

- ***Control statement is a statement whose execution results*** in a choice being made as to which of two or more paths should be followed.
- In other words, the control statements determine the 'flow of control' in a program.

- **Selection statements allow a program to select a particular** execution path from a set of one or more alternatives. Various forms of the **if..else** statement belong to this category.
- **Iteration statements are used to execute a group of one** or more statements repeatedly. **"while, for, and do..while"** statements falls under this group.
- **Jump statements cause an unconditional jump to some** other place in the program. **Goto** statement falls in this group

Program Statements(control)

```
if (expression)
    {
     Block of statements;
     }
else if(expression)
    {
    Block of statements;
     }
else
    {
    Block of statements;
     }
```

```
while ( expression )
{
        Single statement
        or
        Block of statements;
 }

for( expr1; expr2; expr3)
{
        Single statement
        or Block of statements;
}
```

Program Statements(control)

```
do
  {
     Single statement
     or Block of statements;
   }while(expression);
```

Control Statements

Control Statements Include

Selection Statements

- if
- if-else
- switch

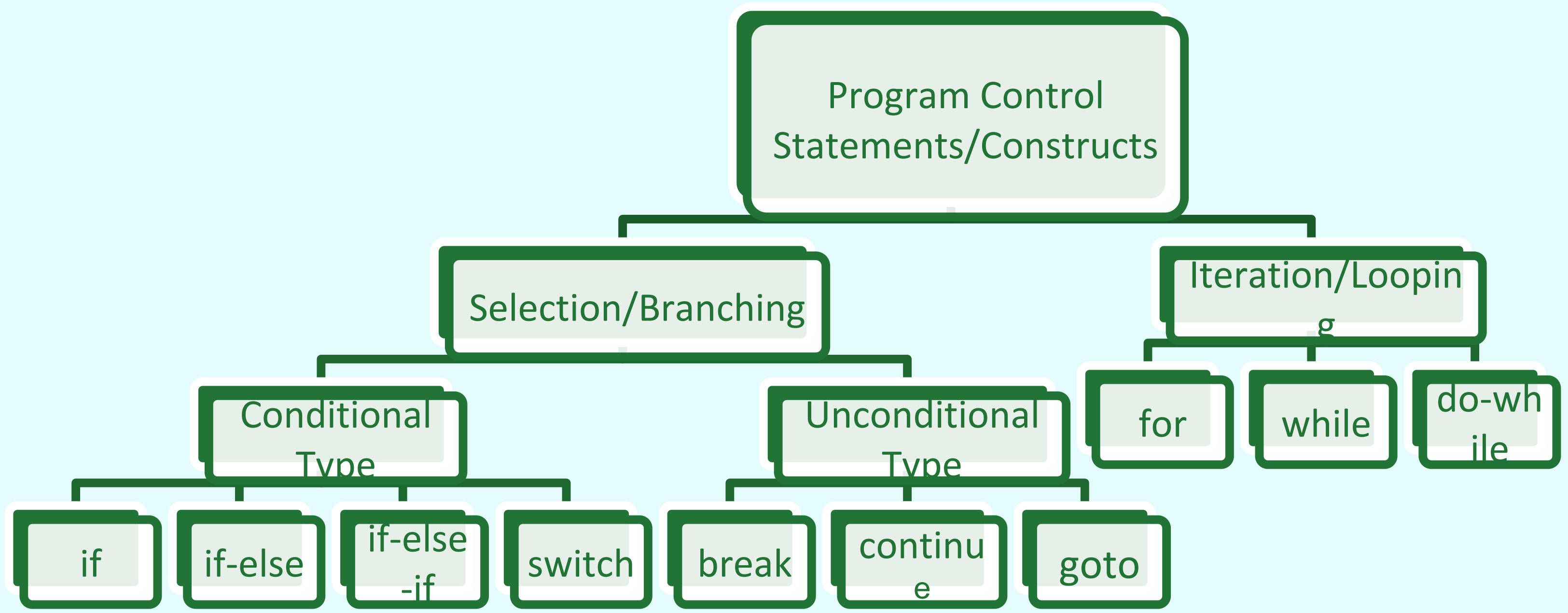
Iteration Statements

- for
- while
- do-while

Jump Statements

- goto
- break
- continue
- return

Program Control Statements/Constructs in 'C'



Relational Operators

To Specify	Symbol Used
less than	<
greater than	>
less than or equal to greater than or equal to	<= >=

Equality and Logical Operators

To Specify	Symbol Used
Equal to	==
Not equal to	!=
Logical AND	&&
Logical OR	||
Negation	!

Points to Note

- If an expression, involving the relational operator, is true, it is given a value of 1. If an expression is false, it is given a value of 0.
- Similarly, if a numeric expression is used as a test expression, any non-zero value (including negative) will be considered as true, while a zero value will be considered as false.
- Space can be given between operand and operator (relational or logical) but space is not allowed between any compound operator like <=, >=, ==, !=. It is also compiler error to reverse them.
- a == b and a = b are not similar, as == is a test for equality, a = b is an assignment operator. Therefore, the equality operator has to be used carefully.
- The relational operators have lower precedence than all arithmetic operators.

A Few Examples

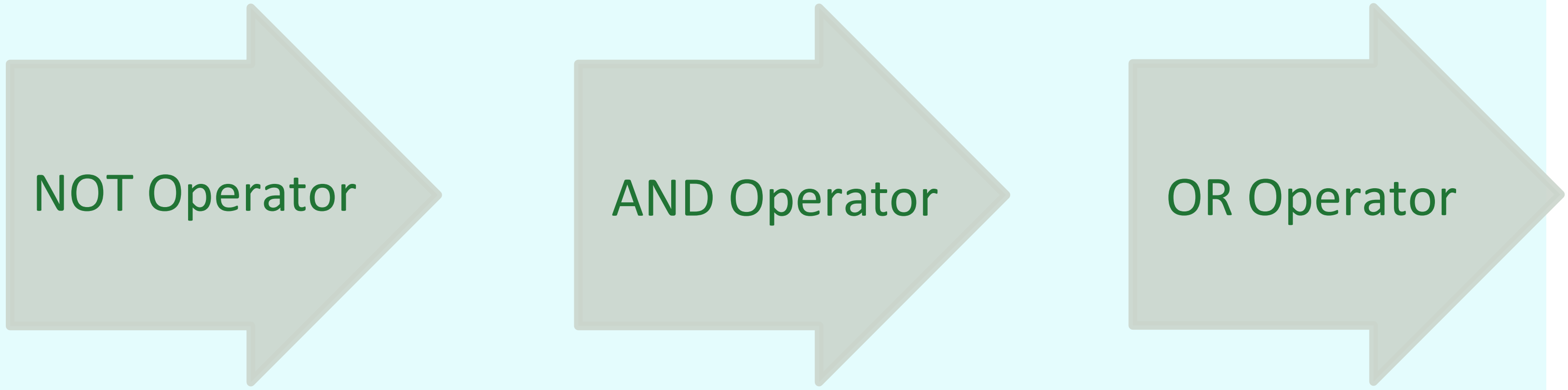
The following declarations and initializations are given:

```
int x=1, y=2, z=3;
```

Then,

- The expression x>=y evaluates to 0 (false).
- The expression x+y evaluates to 3 (true).
- The expression x=y evaluates to 2 (true).

Logical operators may be mixed within relational expressions but their precedence rules must be followed.



Conditional Execution and Selection

- Selection Statements
- The Conditional Operator
- The switch Statement

Selection Statements

One-way decisions using if statement

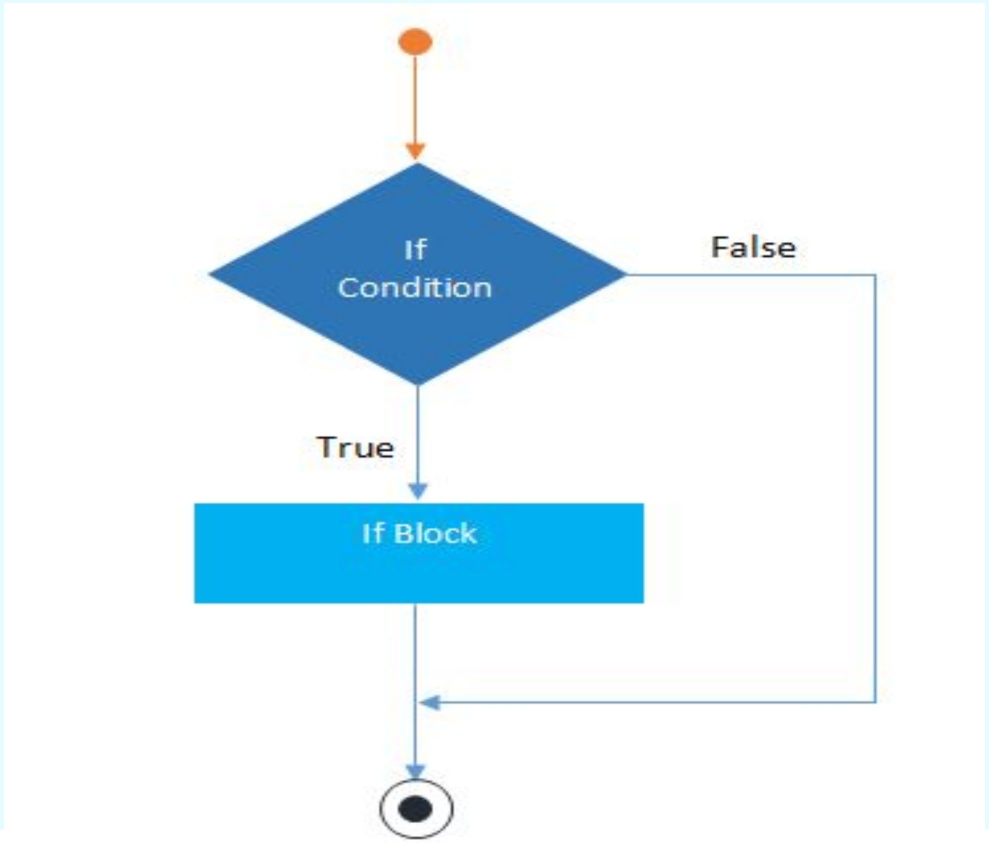
Two-way decisions using if-else statement

Multi-way decisions

Dangling else Problem

One-way decisions using if statement

Flowchart for if construct



If statement

Write a program that prints the number greater than 5.

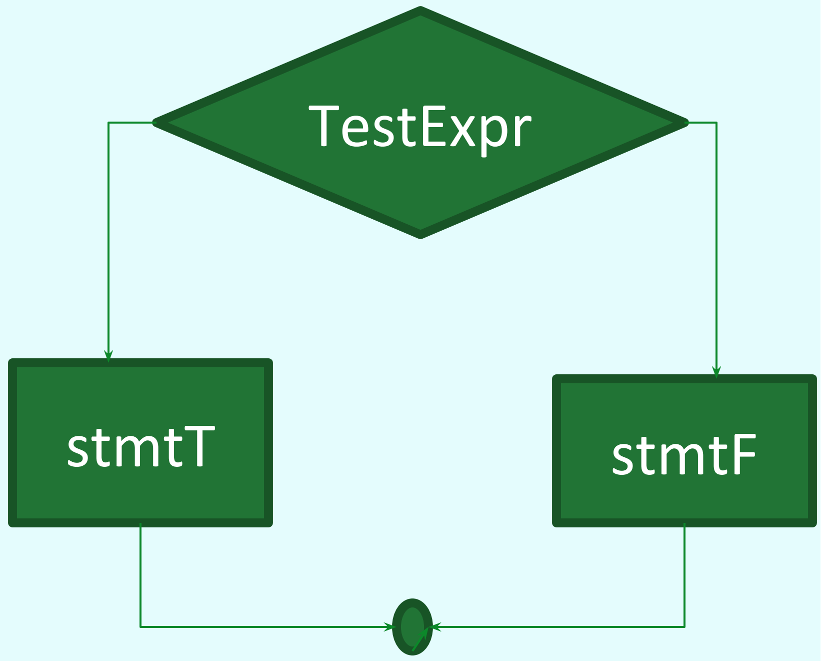
```
#include<stdio.h>
int main()
{
int a;
printf("Enter the value of a \n");
scanf("%d", &a);
if(a>5)
{
printf("The value of a = %d\n",a);
}
printf("Enter a valid number \n");
return 0;
}
```

Two-way decisions using if-else statement

The form of a two-way decision is as follows:

```
if(TestExpr)
  stmtT;
else
  stmtF;
```

Flowchart of if-else construct



Write a program that prints the largest among three numbers.

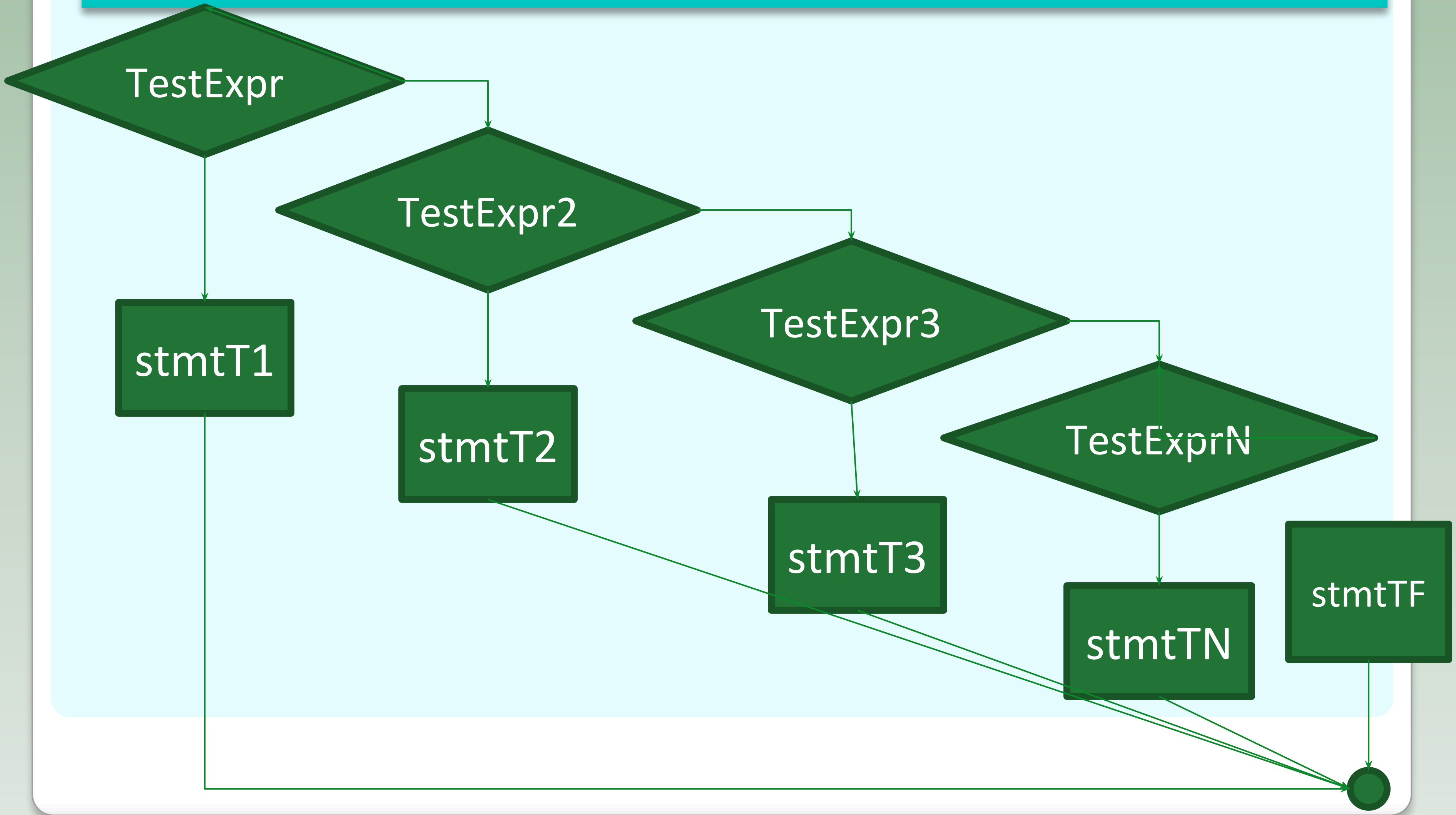
```
#include<stdio.h>
int main()
{
int a;
printf("Enter the value of a \n");
scanf("%d", &a);
if(a>5)
{
printf("The entered value is greater than 5 \n");
}
else
{
printf("The entered value is less than 5 \n");
}
return 0;
}
```

Multi way Decisions

```
if(TestExpr1)
  stmtT1;
    else if(TestExpr2)
      stmtT2;
        else if(TestExpr3)
          stmtT3;
            .. .
              else if(TestExprN)
                 stmtTN;
                   else
                      stmtF;
```

if-else-if ladder

Flowchart of an if-else-if Construct



The following program checks whether a number given by the user is zero, positive, or negative

```
#include<stdio.h>
int main()
{
int a;
printf("Enter the value of a \n");
scanf("%d", &a);
if(a>5)
        printf("The entered value is greater than 5 \n");
else if (a == 5)
        printf("The entered value is equal to 5 \n");
else
        printf("The entered value is less than 5 \n");
return 0;
}
```

Nested if

- When any if statement is written under another if statement, this cluster is called a nested if.
- The syntax for the nested is given here:

Construct 1

```
if(TestExprA)

if(TestExprB)
      stmtBT;
    else
      stmtBF;
 else
    stmtAF;
```

Construct 2

```
if(TestExprA)
   if(TestExprB)
      stmtBT;
   else
      stmtBF;
else

if(TestExprC)
       stmtCT;
    else
       stmtCF;
```

Check whether the number is positive and between 20 and 30.

```
#include<stdio.h>
int main()
{
int a;
printf("Enter the value of a \n");
scanf("%d", &a);
if(a>20)
{
    if (a<30)
      printf("Number is between 20 and 30
     \n");
    else
      printf("Number is greater than 30
     \n");
}
else
{
      if (a>0)
         printf("Number is positive
     and less than 20 \n");
               else
         printf("Number is negative
     \n");
}
return 0;
}
```

Dangling else Problem

- This classic problem occurs when there is no matching else for each if. To avoid this problem, the simple C rule is that always pair an else to the most recent unpaired if in the current block.
- The else is automatically paired with the closest if. But, it may be needed to associate an else with the outer if also.

```
if (condition)
      if (condition)

        if (condition)
   else
            printf("dangling else!\n");
```

Dangling Else

```
#include<stdio.h>
int main()
{
int a;
printf("Enter a number");
scanf("%d",&a);
if (a>10)
if (a<20)
printf("Hyy");
else;
else
printf("exit");
}
```

Solution to Dangling Else Problem

- Use of null else
- Use of braces to enclose the true action of the second if

With null else	With braces
if(TestExprA	if(TestExprA
)	)
	{
if(TestExprB)	
stmtBT;	if(TestExprB)
else	stmtBT;
;	}
else	else
stmtAF;	stmtAF;

Assignment

- WAP to read an alphabet from the user and convert it into uppercase if the entered alphabet is in lowercase, otherwise display an appropriate message.
- Write a program that prints the largest among three numbers using only if statement.
- WAP to test whether a number entered through keyboard is ODD or EVEN using if…..else statement.
- Write a program that prints the largest among three numbers using nested if statement.
- Find out an entered alphabet is vowel or consonant.
- WAP to determine whether a year entered through the keyboard is a leap year or not.

Assignment

WAP using if else statement to create KIIT grade system

- 90-100 is ‘O’ grade
- 80-89 is ‘E’ grade
- 70 to 79 is ‘A’ grade
- 60 to 69 is ‘B’ grade
- 50 to 59 is ‘C’ grade
- 40 to 49 is ‘D’ grade
- below 40 is ‘F’ grade

Write a program that prints the largest among three numbers.

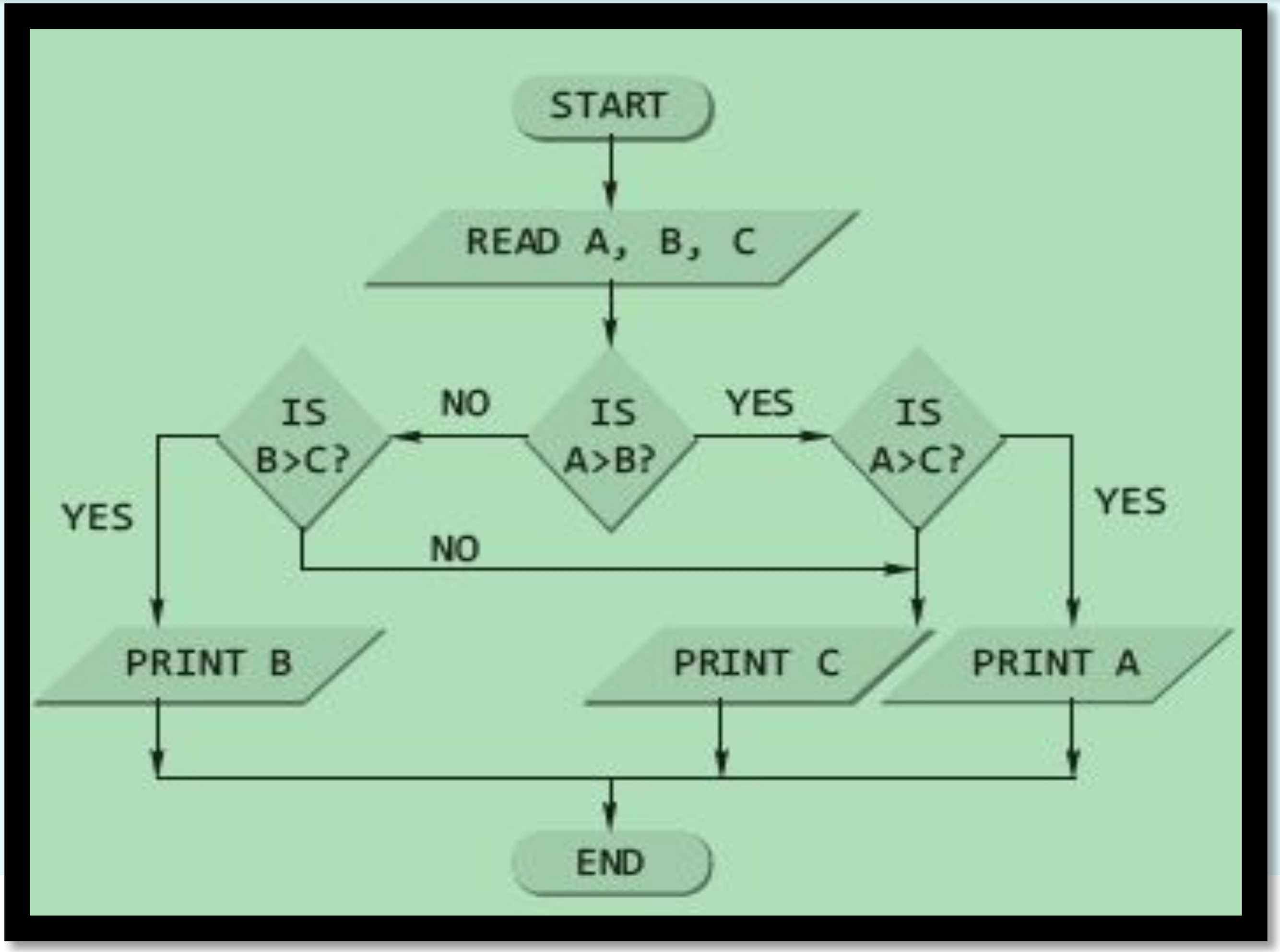
Algorithm

```
1. START
2. PRINT “ENTER THREE NUMBERS”
3. INPUT A, B, C
4. MAX=A
5. IF B>MAX THEN MAX=B
6. IF C>MAX THEN MAX=C
7. PRINT “LARGEST NUMBER IS”, MAX
8. STOP
```

C Program

```c
#include <stdio.h>
int main()
{
int a, b, c, max;
printf(“\nEnter 3 numbers”);
scanf(“%d %d %d”, &a, &b, &c);
max=a;
if(b>max)
{
   max=b;
}
if(c>max)
{
   max=c;
}
printf(“Largest No is %d”, max);
return 0;
}
```

A program to find the largest among three numbers using the nested if



A program to find the largest among three numbers using the nested if

```
#include <stdio.h>
int main()
{
  int a, b, c;
  printf(“\nEnter the three numbers”);
  scanf(“%d %d %d”, &a, &b, &c);
  if(a > b)
  {    if(a > c)
         printf(“%d”, a);
      else
          printf(“%d”, c);
}
 else
 {  if(b > c)
          printf(“%d”, b);
      else
          printf(“%d”, c);
}
  return 0;
}
```

PROGRAM

WAP to read an alphabet from the user and convert it into uppercase if the entered alphabet is in lowercase, otherwise display an appropriate message.

```
#include<stdio.h>
int main()
{
char ch;
printf("\n Enter an alphabet:");
scanf("%c", &ch);
if (ch>='a' && ch<='z')
{
ch=ch-32;
printf("\n The uppercase of the entered alphabet is %c", ch);
}
else
printf("\nThe entered character is not a lower case alphabet");
return 0;
}
```

PROGRAM
Find out an entered alphabet is vowel or consonant

```
If entered character is alphabet (small or
capital range of alphabets)
    If ch is vowel (a, e, i, o, u both in capital or small)
        print vowel
    else
        print conso

else
enter valid alphabets
```

```
#include<stdio.h>
int main()
{
char ch;
printf("\n Enter an alphabet: ");
scanf("%c", &ch);
if ((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
{
    if (ch=='a' || ch=='A' || ch=='e' || ch=='E' || ch=='i' || ch=='I' || ch=='o' ||
ch=='O' || ch=='u' ||ch=='U')
        printf("\nThe entered character %c is a vowel", ch);
    else
        printf("\nThe entered character %c is a consonant", ch);
}
else
    printf("\nThe entered character %c is not an alphabet",ch);

return 0;
}
```

WAP to determine whether a year entered through the keyboard is a leap year or not.

```
#include<stdio.h>
int main()
{
int year;
printf("\nEnter the year:");
scanf("%d",&year);
if((year%4==0 && year%100!=0)||(year %400==0))
    printf("\n%d is a leap year.",year);
else
    printf("\n%d is not a leap year.",year);
return 0;
}
```

The Conditional Operator

- It has the following simple format:

  expr1 ? expr2 : expr3

  It executes by first evaluating expr1, which is normally a relational expression, and then evaluates either expr2, if the first result was true, or expr3, if the first result was false.

```
#include <stdio.h>
int main()
{
 int a,b,c;
 printf("\n ENTER THE TWO
    NUMBERS:");
  scanf("%d %d", &a, &b);

  c=a>b? a : b>a ? b :-1;

  if(c==-1)
        printf("\n BOTH NUMBERS ARE
    EQUAL");
  else
  printf("\n LARGER NUMBER IS %d",c);
  return 0;
}
```

An Example