Control Flow: Branching

CS10001: Programming & Data Structures

Sudeshna Sarkar
Professor, Dept. of Computer Sc. & Engg.,
Indian Institute of Technology Kharagpur

Statements and Blocks

An expression followed by a semicolon becomes a statement.

```
x = 5;
i++;
printf (“The sum is %d\n”, sum”) ;
```

Braces { and } are used to group declarations and statements together into a compound statement, or block.

```
{
    sum = sum + count;
    count++;
    printf (“sum = %d\n”, sum) ;
}
```

Control Statements: What do they do?

- **Branching:**
  - Allow different sets of instructions to be executed depending on the outcome of a logical test.
    - Whether TRUE (non-zero) or FALSE (zero).

- **Looping:**
  - Some applications may also require that a set of instructions be executed repeatedly, possibly again based on some condition.

How do we specify the conditions?

- **Using relational operators.**
  - **Four relation operators:** **<, <=, >, >=**
  - **Two equality operations:** **==, !=**

- **Using logical operators / connectives.**
  - **Two logical connectives:** **&&, | |**
  - **Unary negation operator:** **!**

Expressions

(count <= 100)

((math+phys+chem)/3 >= 60)

((sex == 'M') && (age >= 21))

((marks >== 80) && (marks < 90))

((balance > 5000) | | (no_of_trans > 25))

(! (grade == 'A'))

The conditions evaluate to …

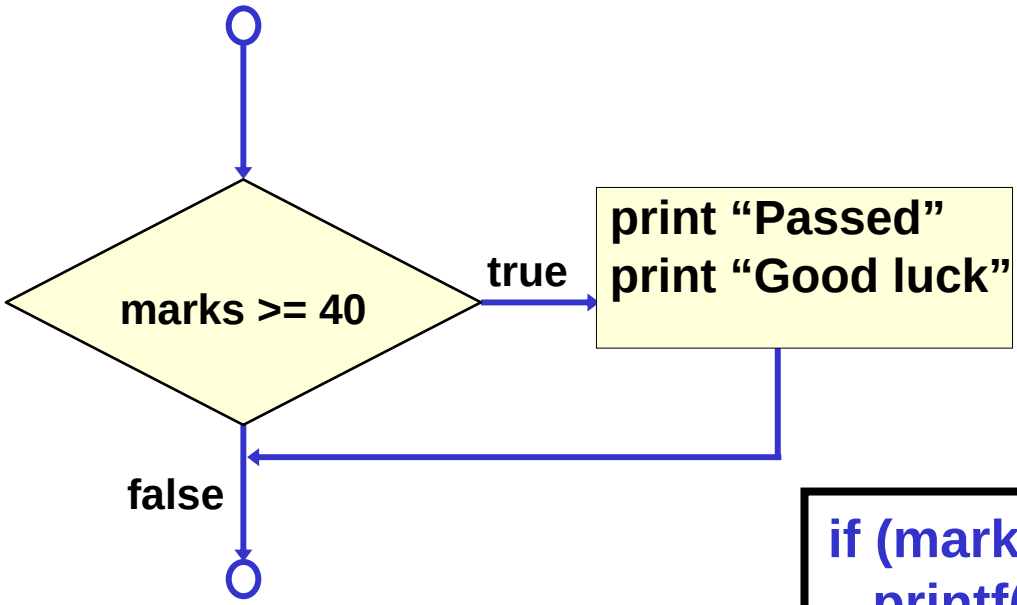
- **Zero**
  - **Indicates FALSE.**
- **Non-zero**
  - **Indicates TRUE.**
  - **Typically the condition TRUE is represented by the value ‘1’.**

Branching: *The if Statement*

```
if (expression)
        statement;

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

The condition to be tested is any expression enclosed in parentheses. The expression is evaluated, and if its value is non-zero, the statement is executed.



A decision can be made on any expression.

zero - false

nonzero - true

```
if (marks>=40)  {
   printf(“Passed \n”);
   printf(“Good luck\n”);
}
printf (“End\n”) ;
```

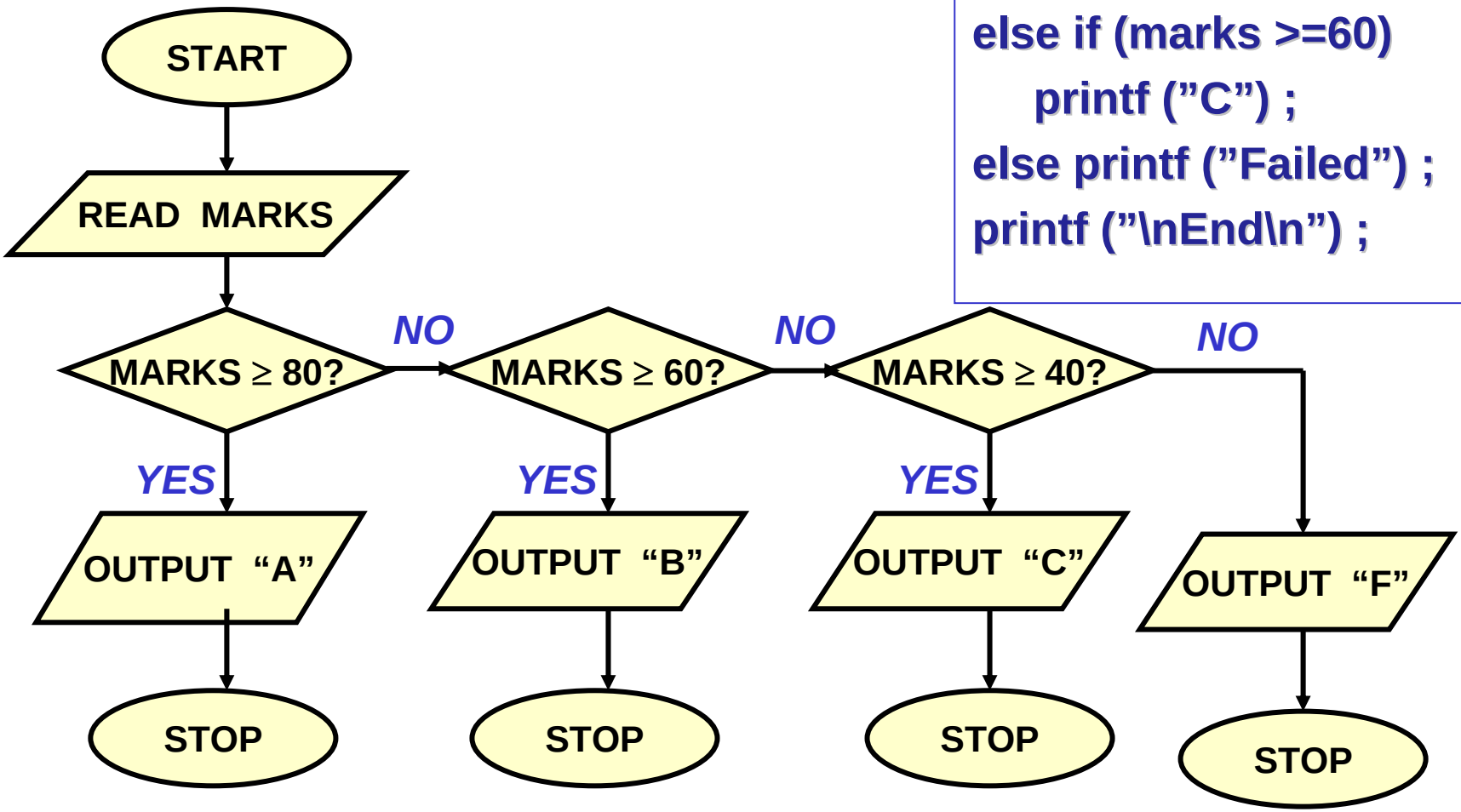
Branching: *if-else Statement*

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

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

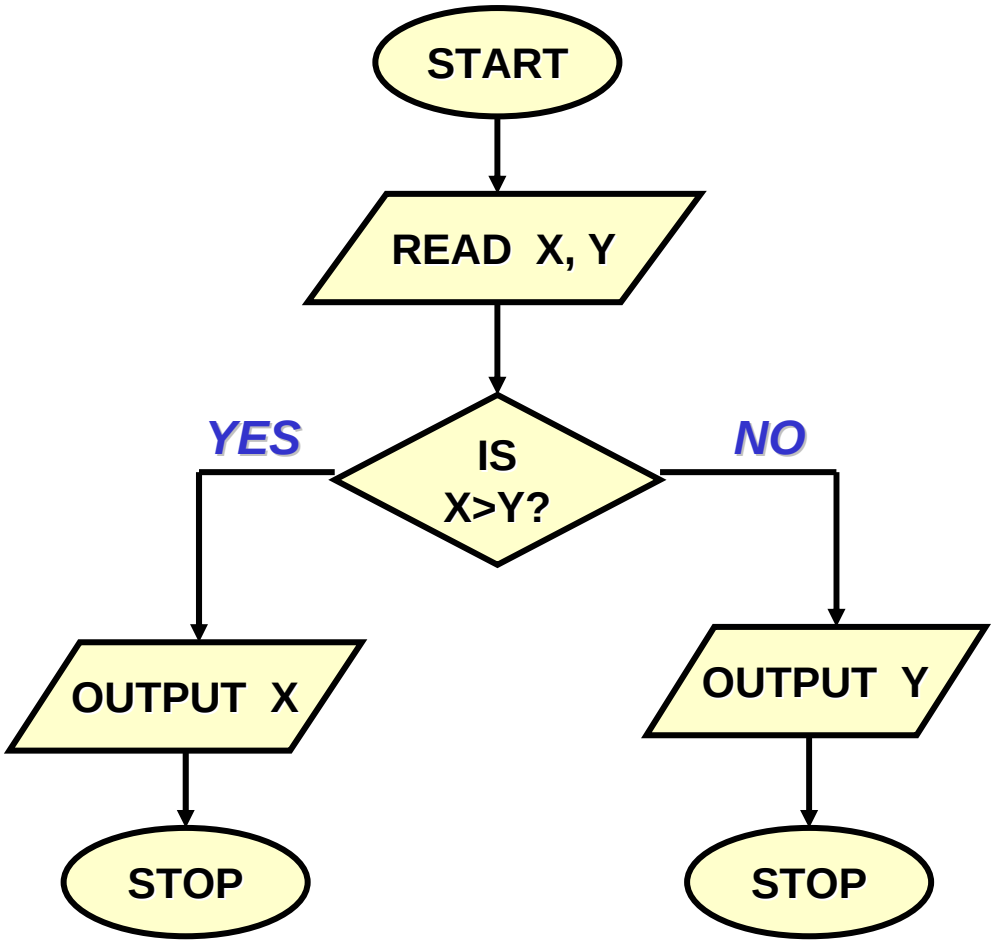
Grade Computation

```
if (marks >= 80)
    printf (”A”) ;
else if (marks >= 60)
    printf (”B”) ;
else if (marks >=60)
    printf (”C”) ;
else printf (”Failed”) ;
printf (”\nEnd\n”) ;
```



```
int main () {
      int marks;
     scanf (“%d”, & marks) ;
      if (marks>= 80) {
          printf (“A”) ;
          printf (“Good Job!”) ;
      }
      else if (marks >= 60)
          printf (“B”) ;
      else if (marks >=60)
          printf (“C”) ;
      else {
          printf (“Failed”) ;
          printf (“Study hard for the supplementary”) ;
      }
      printf (“\nEnd\n”) ;
}
```

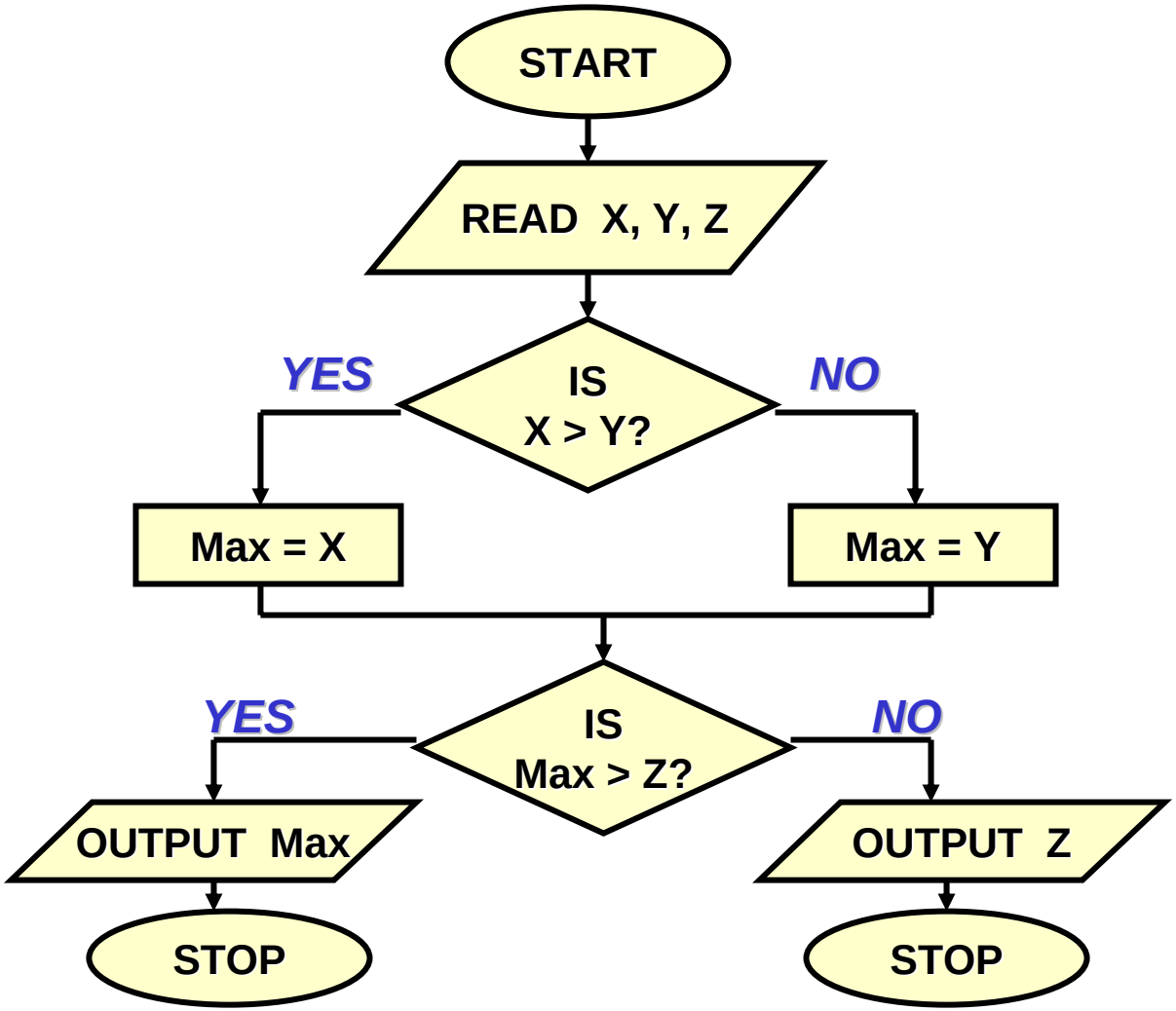
Find the larger of two numbers

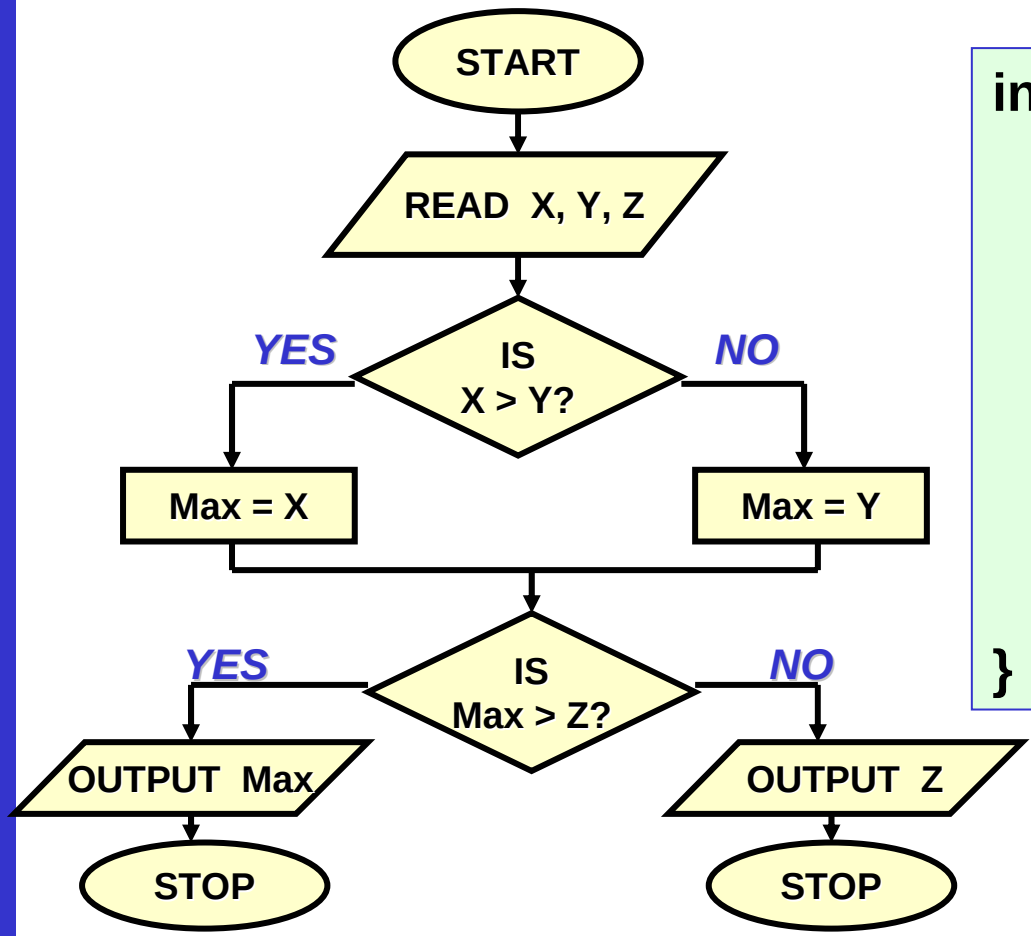



```
int main () {
    int x, y;

    scanf ("%d%d", &x, &y) ;
    if (x>y)
        printf ("%d\n", x);
    else
        printf ("%d\n", x);
}
```

Example 3: *Largest of three numbers*






```
int main () {
      int x, y, z, max;
      scanf ("%d%d%d",&x,&y,&z);
      if (x>y)
            max = x;
      else max = y;
      if (max > z)
            printf ("%d", max) ;
      else printf ("%d",z);
}
```

Example

```
#include <stdio.h>
main()
{
    int  a,b,c;
    scanf (“%d %d %d”, &a, &b, &c);
    if ((a>=b) && (a>=c))
        printf (“\n The largest number is: %d”, a);
    if ((b>=a) && (b>=c))
        printf (“\n The largest number is: %d”, b);
   if ((c>=a) && (c>=b))
        printf (“\n The largest number is: %d”, c);
}
```

Confusing Equality (==) and Assignment (=) Operators

- **Dangerous error**
  - Does not ordinarily cause syntax errors.
  - Any expression that produces a value can be used in control structures.
  - Nonzero values are true, zero values are false.

- **Example:**

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );

if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

Nesting of if-else Structures

- It is possible to nest if-else statements, one within another.
- All “if” statements may not be having the “else” part.
  - Confusion??
- Rule to be remembered:
  - An “else” clause is associated with the closest preceding unmatched “if”.
  - Some examples shown next.

Dangling else problem

if (exp1) if (exp2) stmta else stmtb

```
if (exp1) {
    if (exp2)
        stmta
    else
        stmtb
}
```

OR

```
if (exp1) {
    if (exp2)
        stmta
}
else
    stmtb
```

?

Which one is the correct interpretation?

Dangling else problem

if (exp1) if (exp2) stmta else stmtb

```
if (exp1) {
    if (exp2)
        stmta
    else
        stmtb
}
```

More examples

```
if e1 s1
else if e2 s2
```

```
if e1 s1
else if e2 s2
else s3
```

```
if e1 if e2 s1
else s2
else s3
```

```
if e1 if e2 s1
else s2
```

?

Answers

```
if e1 s1
else if e2 s2
```

```
if  e1  s1
else  { if  e2  s2 }
```

```
if e1 s1
else if e2 s2
else s3
```

```
if  e1  s1
else  { if  e2  s2
        else s3 }
```

```
if e1 if e2 s1
else s2
else s3
```

```
 if  e1  { if  e2  s1
           else  s2 }
else  s3
```

```
if e1 if e2 s1
else s2
```

```
if  e1  { if  e2  s1
          else s2 }
```

Common Errors

```
c = getchar( );
if ((c == 'y') && (c == 'Y')) printf("Yes\n");
else printf("No\n");
```

```
c = getchar( );
if ((c != 'n') || (c != 'N')) printf("Yes\n");
else printf("No\n");
```

The Conditional Operator ?:

- This makes use of an expression that is either true or false. An appropriate value is selected, depending on the outcome of the logical expression.

- Example:

```
interest = (balance>5000) ? balance*0.2 : balance*0.1;
```

Returns a value

Equivalent to:

```
if (balance > 5000)
        interest = balance*0.2;
else interest = balance*0.1;
```

More examples

- **Examples:**

```
x = ((a>10) && (b<5)) ? a+b : 0

(marks>=60) ? printf("Passed \n") : printf("Failed \n");
```

The *switch* Statement

- This causes a particular group of statements to be chosen from several available groups.
  - Uses “switch” statement and “case” labels.
  - Syntax of the “switch” statement:

```
switch (expression)  {
    case expression-1: { …….. }
    case expression-2: { …….. }

    case expression-m: { …….. }
    default: { …….. }
}
```

where “expression” evaluates to int or char

Examples

```
switch ( letter ) {
    case 'A':
        printf ("First letter \n");
        break;
    case 'Z':
        printf ("Last letter \n");
        break;
    default :
        printf ("Middle letter \n");
        break;
}
```

Will print this statement for all letters other than A or Z

Examples

```
switch (choice = getchar()) {
    case ‘r’ :
    case ‘R’: printf(“Red”);
              break;
    case ‘b’ :
    case ‘B’ : printf(“Blue”);
               break;
    case ‘g’ :
    case ‘G’: printf(“Green”);
              break;
    default: printf(“Black”);
}
```

Since there isnt a **break** *statement here, the control passes to the next statement* (printf) ***without*** *checking the next condition.*

Another way

```
switch  (choice = toupper(getchar()))  {

    case ‘R’:       printf (“RED \n”);
                    break;
    case ‘G’:       printf (“GREEN \n”);
                    break;
    case ‘B’:       printf (“BLUE \n”);
                    break;
    default:        printf (“Invalid choice \n”);

}
```

Rounding a Digit

```
switch (digit)   {
        case 0:
        case 1:
        case 2:
        case 3:
        case 4: result = 0; printf (“Round down\n”); break;
        case 5:
        case 6:
        case 7:
        case 8:
        case 9: result = 10; printf(“Round up\n”); break;
}
```

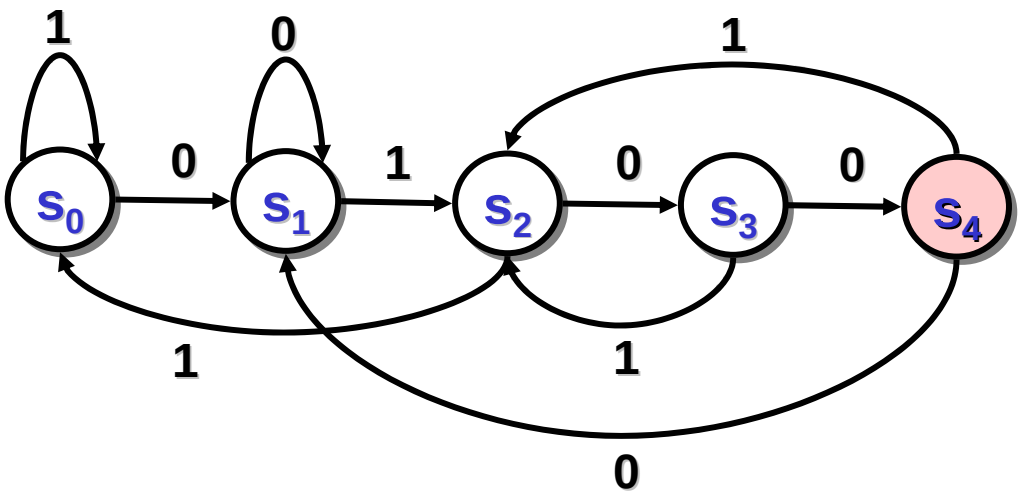
```
int main () {
    int operand1, operand2;
    int result = 0;
    char operation ;
    /* Get the input values */
    printf (“Enter operand1 :”);
    scanf(“%d”,&operand1) ;
    printf (“Enter operation :”);
    scanf (“\n%c”,&operation);
    printf (“Enter operand 2 :”);
    scanf (“%d”, &operand2);
    switch (operation)    {
    case ‘+’ :
        result=operand1+operand2;
          break;
    case ‘-’ :
        result=operand1-operand2;
        break;
    case ‘*’ :
        result=operand1*operand2;
        break;
    case ‘/’ :
        if (operand2 !=0)
            result=operand1/operand2;
        else
            printf(“Divide by 0 error”);
        break;
    default:
        printf(“Invalid operation\n”);
    }
    printf (“The answer is %d\n”,result);
}
```

The *break* Statement

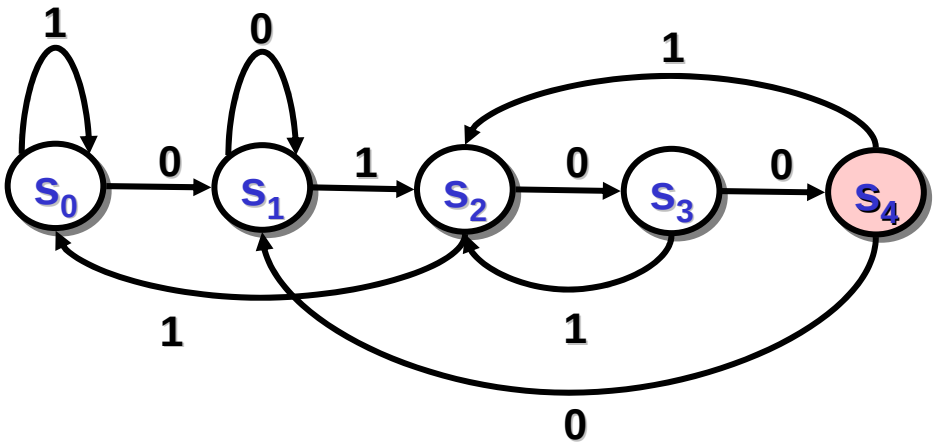
- Used to exit from a switch or terminate from a loop.
- With respect to “switch”, the “break” statement causes a transfer of control out of the entire “switch” statement, to the first statement following the “switch” statement.
- Can be used with other statements also …

Example: *Pattern Matching*

- Write a program that reads an arbitrarily long sequence of bits terminated with a null character and counts the number of occurrences of the sequence “0100”



Example: *Pattern Matching*




```
c = getchar( );
switch (state) {
   case 0: if (c == '0') state = 1;
           else state = 0; break;
   case 1: if (c == '0') state = 1;
           else state = 2; break;
   case 2: if (c == '0') state = 3;
           else state = 0; break;
   case 3: if (c == '0') state = 4;
           else state = 2;
           count++; break;
   case 4: if (c == '0') state = 1;
           else state = 2; break;
}
```

Example: *Pattern Matching*

```
c = getchar( ); count = 0; state = 0;
while (c != '\0') {
        switch (state) {
          case 0: if (c == '0') state = 1;
                  else state = 0; break;
           --------
           --------
          case 3: if (c == '0') state = 4;
                  else state = 2;
                  count++; break;
          case 4: if (c == '0') state = 1;
                  else state = 2; break;
        }
        c = getchar( );
}
printf("No of matches: %d\n", count);
```

A Look Back at Arithmetic Operators:

The Increment and Decrement

Increment (++) and Decrement (--)

- Both of these are unary operators; they operate on a single operand.
- The increment operator causes its operand to be increased by 1.
  - Example: a++, ++count
- The decrement operator causes its operand to be decreased by 1.
  - Example: i--, --distance

Pre-increment versus post-increment

- **Operator written before the operand (++i, --i))**
  - **Called pre-increment operator.**
  - **Operator will be altered in value *before* it is utilized for its intended purpose in the program.**

- **Operator written after the operand (i++, i--)**
  - **Called post-increment operator.**
  - **Operator will be altered in value *after* it is utilized for its intended purpose in the program.**

Examples

Initial values :: a = 10; b = 20;

x = 50 + ++a;	a = 11, x = 61
x = 50 + a++;	x = 60, a = 11
x = a++ + --b;	b = 19, x = 29, a = 11
x = a++ – ++a;	??

Called side effects:: while calculating some values, something else get changed.