Conditional Statement

Conditional Statements

- Allow different sets of instructions to be executed depending on truth or falsity of a logical condition
- Also called Branching
- How do we specify conditions?
  - Using expressions
    - non-zero value means condition is true
    - value 0 means condition is false
  - Usually logical expressions, but can be any expression
    - The value of the expression will be used

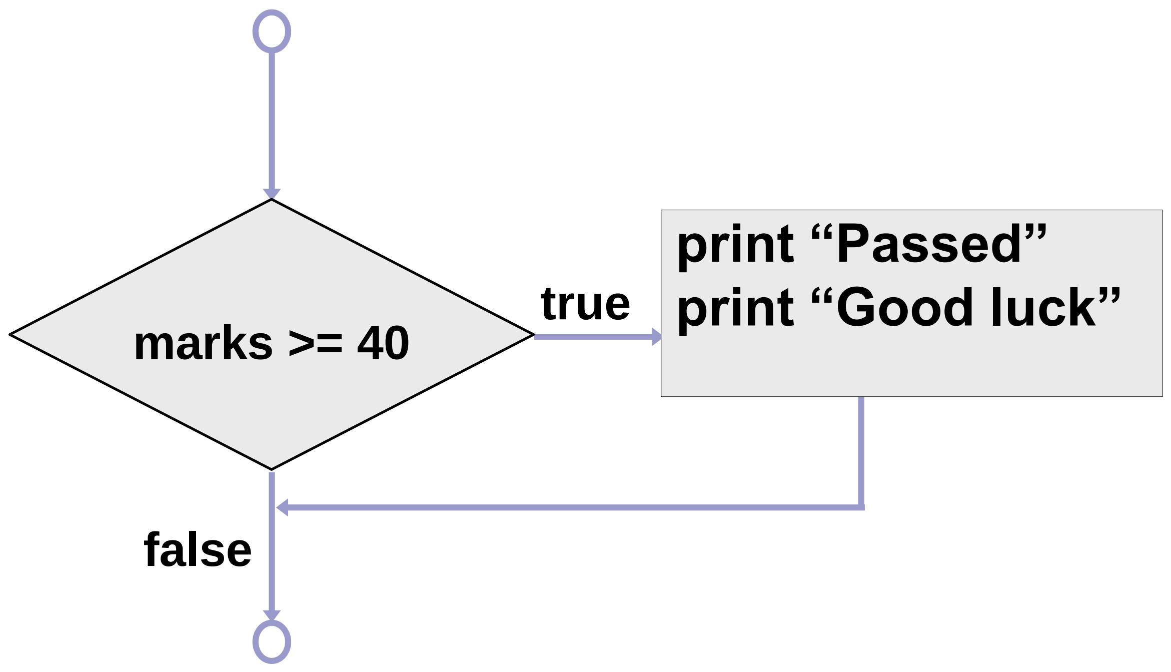
Branching: if Statement

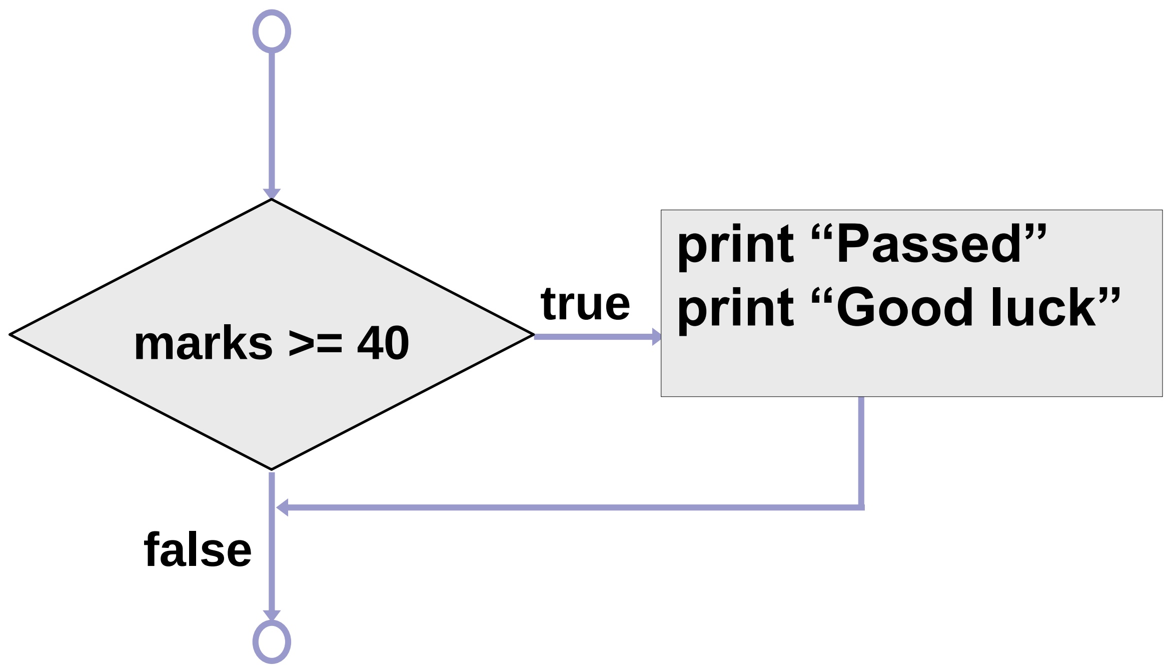
```
if (expression)
        statement;

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

Branching: 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.


marks >= 40
true
print “Passed”
print “Good luck”
false

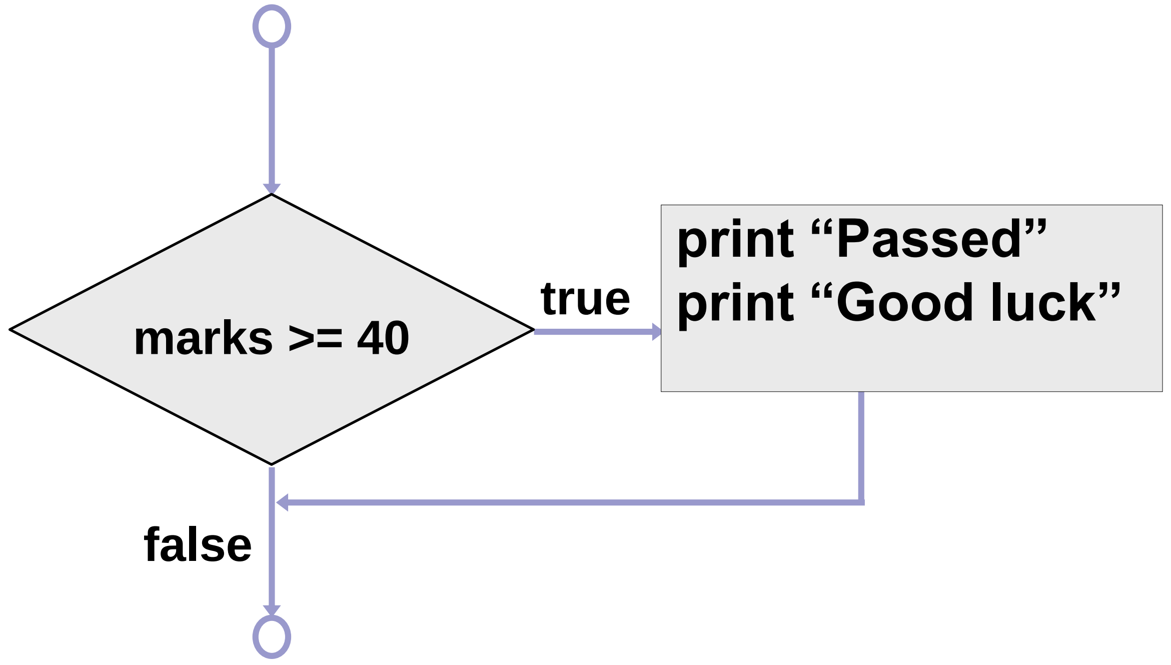


print “Passed”
print “Good luck”

A decision can be made on any expression.

zero - false

nonzero - true



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;
}
```

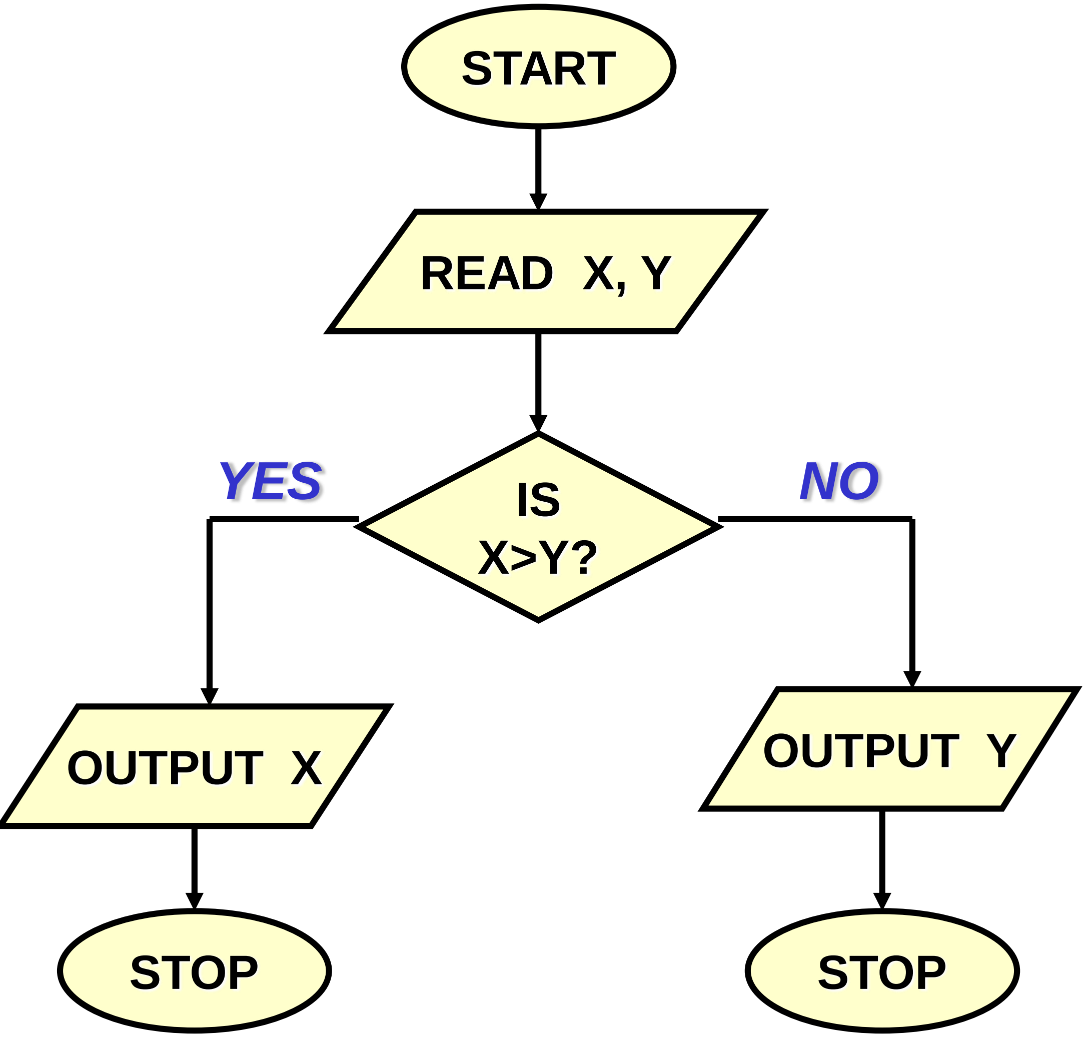
```
if (expression) {
    Block of statements;
}
else if  (expression) {
    Block of statements;
}
else {
    Block of statements;
}
```

Grade Computation

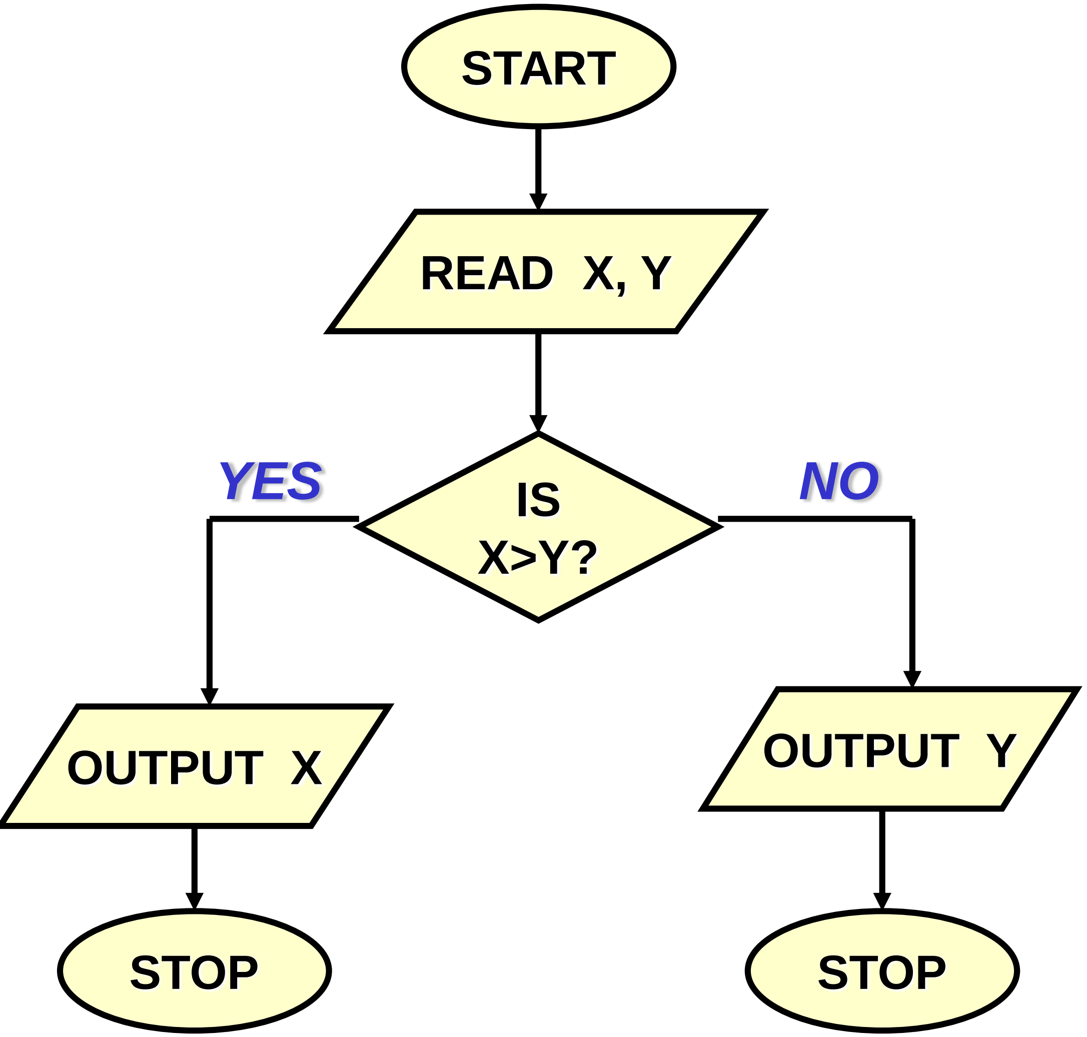
```
void main()  {
    int marks;
    scanf(“%d”, &marks);
    if (marks >= 80)
         printf (”A”) ;
   else if (marks >= 70)
         printf (”B”) ;
    else if (marks >= 60)
         printf (”C”) ;
     else printf (”Failed”) ;
}
```

```
void main () {
     int marks;
     scanf (“%d”, &marks) ;
     if (marks>= 80) {
        printf (“A: ”) ;
        printf (“Good Job!”) ;
     }
     else if (marks >= 70)
        printf (“B ”) ;
     else if (marks >= 60)
        printf (“C ”) ;
     else {
        printf (“Failed: ”) ;
        printf (“Study hard for the supplementary”) ;
     }
}
```

Find the larger of two numbers

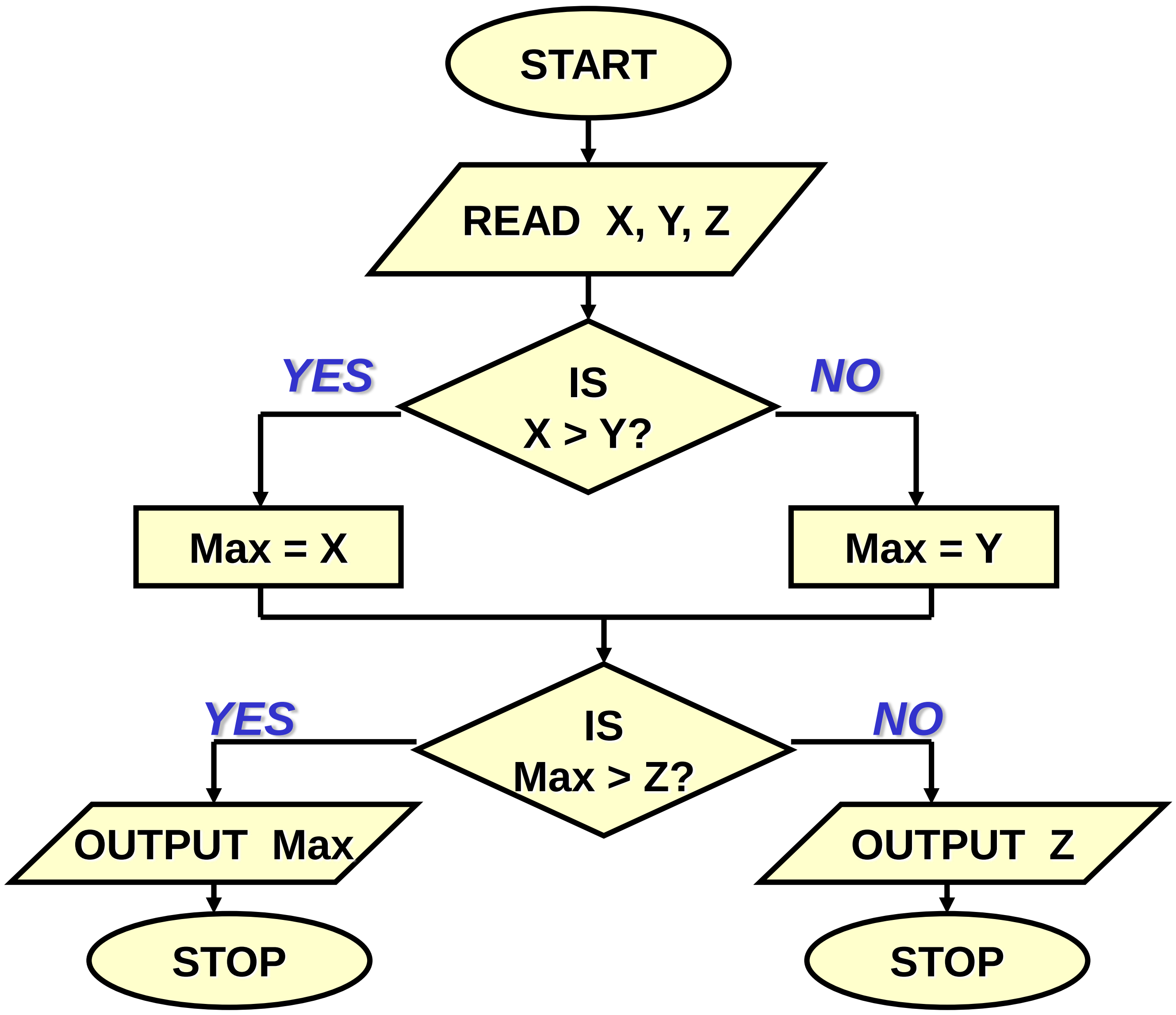


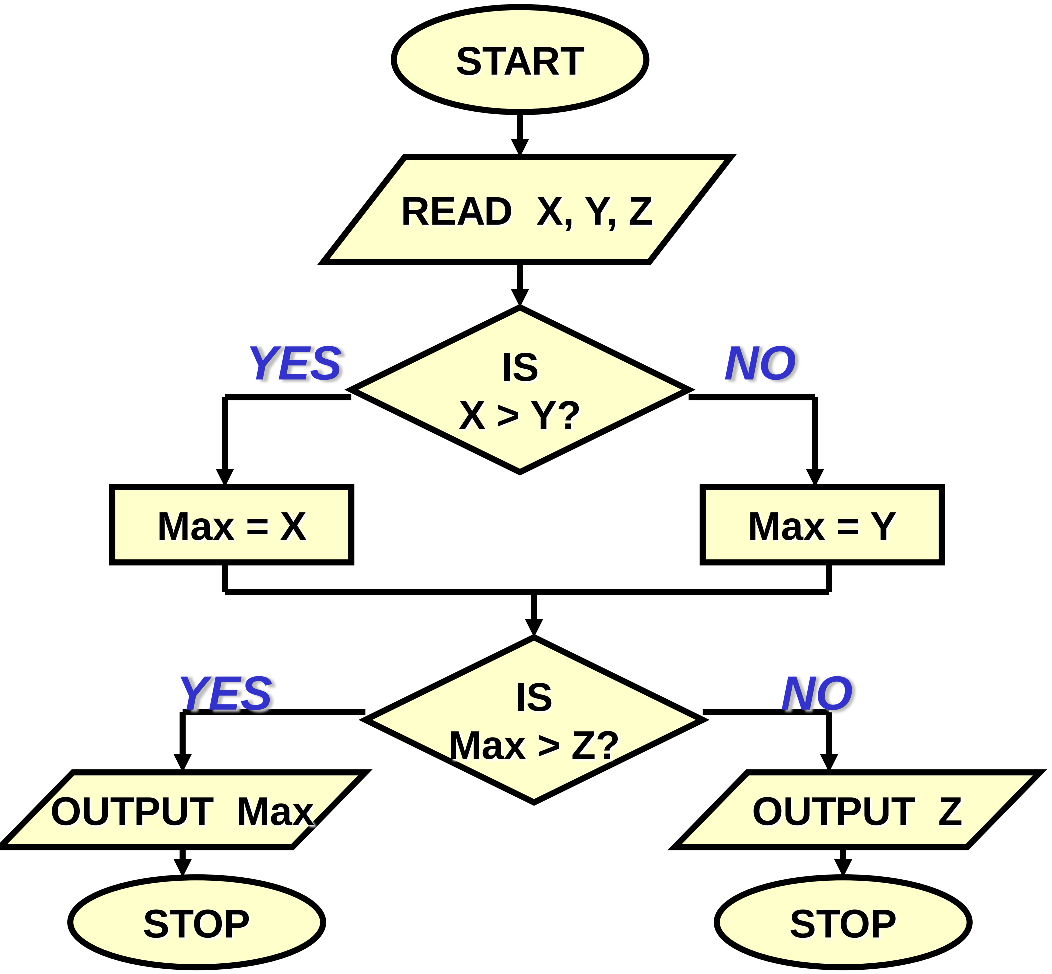
Find the larger of two numbers

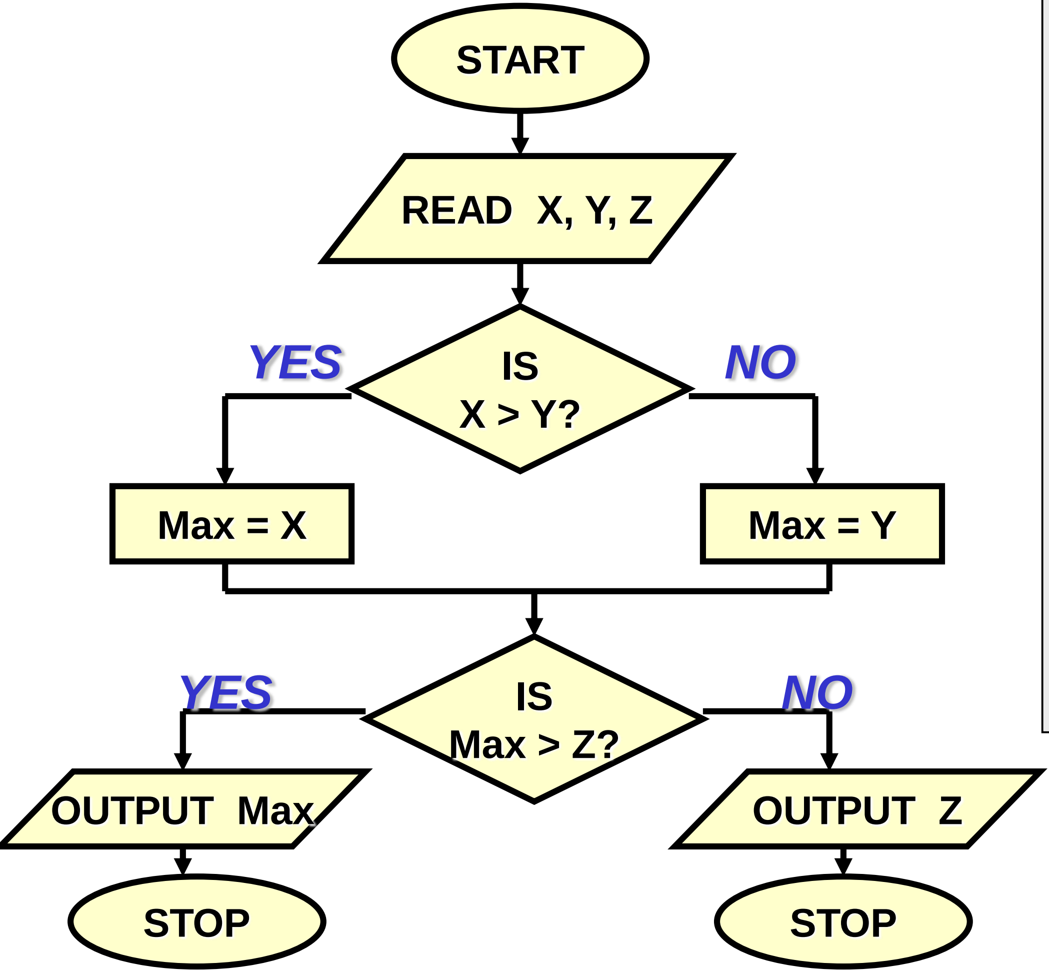


```
void main () {
    int x, y;
    scanf (“%d%d”, &x,
    &y) ;
    if (x > y)
        printf (“%d\n”, x);
    else
        printf (“%d\n”, y);
}
```

Largest of three numbers




START
READ X, Y, Z
IS X > Y?
YES
NO
Max = X
Max = Y
IS Max > Z?
YES
NO
OUTPUT Max
OUTPUT Z
STOP
STOP



```
void 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);
}
```

Another version

```
void 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:

WRONG! Will always print the line

```
    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"

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?

Give braces explicitly in your programs to match the else with the correct if to remove any ambiguity

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
```

?

Answers

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

if e1 s1                         if  e1  s1
else if e2 s2         ==>        else  { if  e2  s2
else s3                                      else s3 }

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

The Conditional Operator ?:

- This makes use of an expression that is either non-0 or 0. An appropriate value is selected, depending on the value of the expression
- Example: instead of writing

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

We can just write

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

More Examples

-
  ```
  if (((a >10) && (b < 5))
      x = a + b;
  else x = 0;

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

-
  ```
  if (marks >= 60)
      printf(“Passed \n”);
  else printf(“Failed \n”);

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

The switch Statement

- An alternative to writing lots of if-else in some special cases
- This causes a particular group of statements to be chosen from several available groups based on equality tests only
- Uses switch statement and case labels

- Syntax

```
switch (expression)  {
   case const-expr-1: S-1
   case const-expr-2: S-2
        :
   case const-expr-m: S-m
   default: S
}
```

- expression is any integer-valued expression
- const-expr-1, const-expr-2,…are any constant integer-valued expressions
  - Values must be distinct
- S-1, S-2, …,S-m, S are statements/compound statements
- Default is optional, and can come anywhere (not necessarily at the end as shown)

Behavior of switch

- expression is first evaluated
- It is then compared with const-expr-1, const-expr-2,…for equality in order
- If it matches any one, all statements from that point till the end of the switch are executed (including statements for default, if present)
  - Use break statements if you do not want this (see example)
- Statements corresponding to default, if present, are executed if no other expression matches

Example

```
int x;
scanf(“%d”, &x);
switch (x) {
     case 1: printf(“One\n”);
    case 2: printf(“Two\n”);
    default: printf(“Not one or two\n”);
};
```

If x = 1 is entered, this will print

One
Two
Not one or two

Not what we want

switch-1.c

Correct Program

```
int x;
scanf(“%d”, &x);
switch (x) {
    case 1: printf(“One\n”);
                break;
    case 2: printf(“Two\n”);
                break;
    default: printf(“Not one or two\n”);
};
```

If x = 1 is entered, this will print

```
One
```

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;
}
```

Since there isn’t a break statement here, the control passes to the next statement without checking the next condition.

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 …(will show later)