Ch4: Making Decisions

- Relational Operators
- The if Statement
- The if/else Statement
- Nested if Statements
- The if/else if Statement
- Flags
- Logical Operators
- Short Circuit Evaluation
- Conditional Operator
- The switch Statement
- Blocks and Variable Scope

Relational Operators

- binary operators compare operands to yield one of two results
  - FALSE or 0
  - TRUE or 1 *

*True is stored **internally** as 1; however, ***any non-zero*** number is considered true.

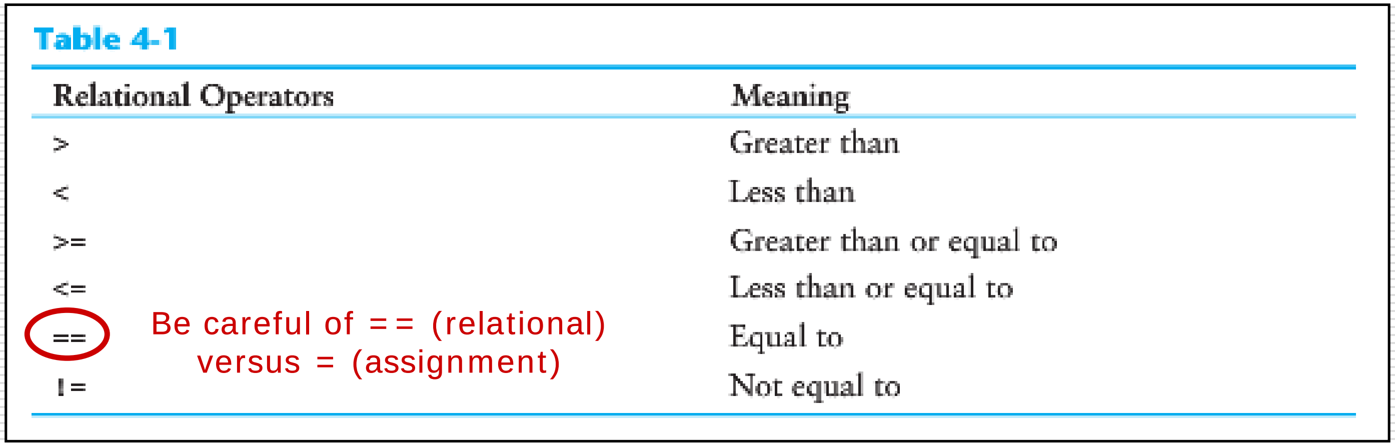


Table 4-1

Relational Operators	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Be careful of == (relational) versus = (assignment)

Relational Operators

- also used to compare characters and string objects

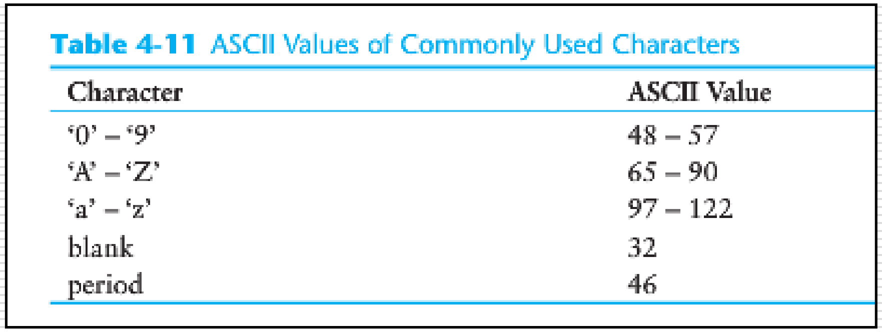


Table 4-11 ASCII Values of Commonly Used Characters

Character	ASCII Value
'0' – '9'	48 – 57
'A' – 'Z'	65 – 90
'a' – 'z'	97 – 122
blank	32
period	46

```
'0' < '9' → true
'z' > 'a' → true
```

$\boxed{48} < \boxed{57}$

$\boxed{122} > \boxed{97}$

Relational Operators

- also used to compare characters and string objects

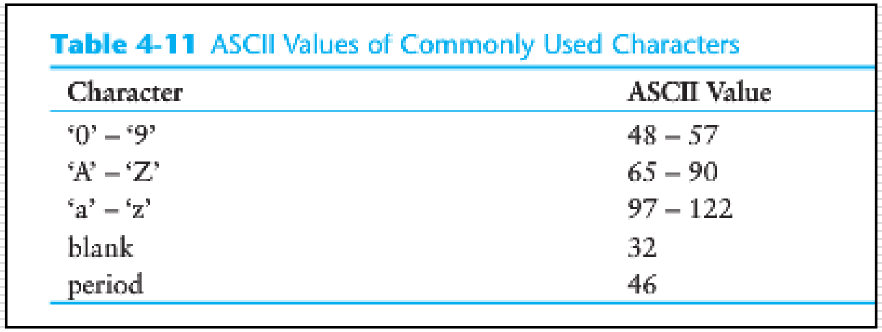


Table 4-11 ASCII Values of Commonly Used Characters

Character	ASCII Value
'0' – '9'	48 – 57
'A' – 'Z'	65 – 90
'a' – 'z'	97 – 122
blank	32
period	46

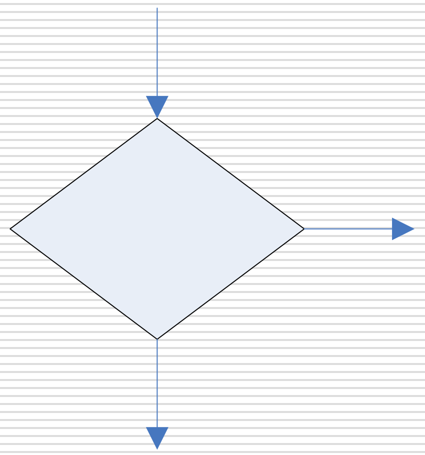
```
"ABC" < "XYZ"
"Mary" > "Mark" → true
"Mary" < "Mark" → false
"Mary" != "Mark" → true
```

$\boxed{65}\boxed{66}\boxed{67} < \boxed{88}\boxed{89}\boxed{90}$

$\boxed{77}\boxed{97}\boxed{114}\boxed{121}\ ?\ \boxed{77}\boxed{97}\boxed{114}\boxed{107}$

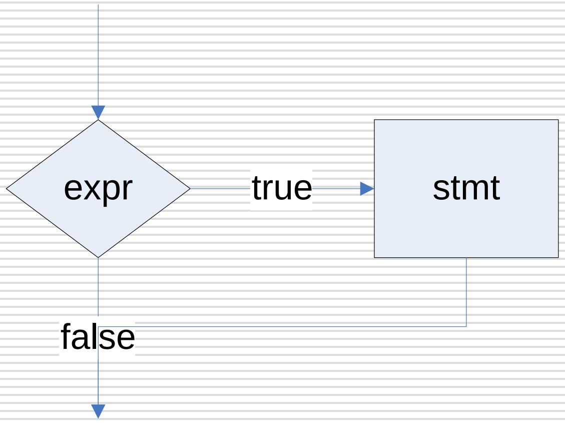
Selection Statements

- selection flow of control statements
  - program must choose between alternative actions
  - selection based upon evaluation of an expression
    - **all** expressions have value and data type
- statements
  - if
  - if/else
  - switch



The if Statement

- **expr** must have arithmetic or pointer type
  - can even be a single number or variable
- if **expr** evaluates to nonzero (that is, TRUE) then **stmt** is executed
- **stmt** can also be a group of statements enclosed within braces { }



```
if (expr)
    stmt;
```

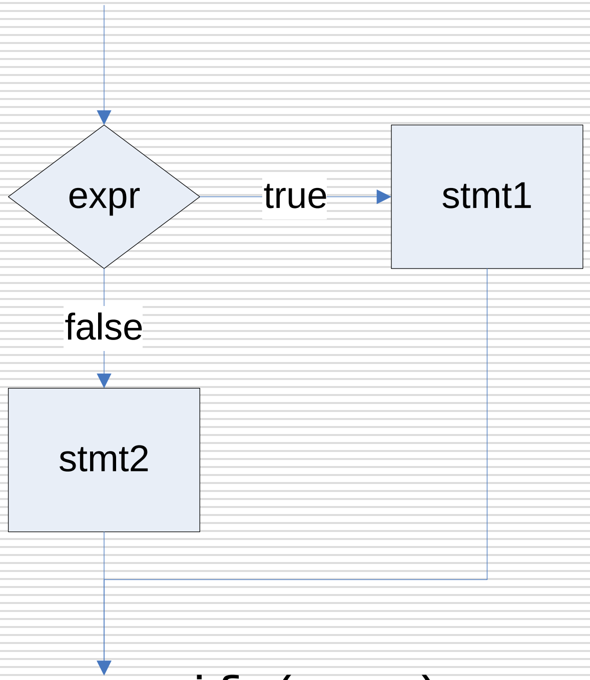
```
if (expr)
{
    stmt1;
    stmt2;
}
```

```
~~if (expr);~~
~~    stmt;~~
```

```
~~if (expr)~~
~~    stmt1;~~
~~    stmt2;~~
```

The if/else Statement

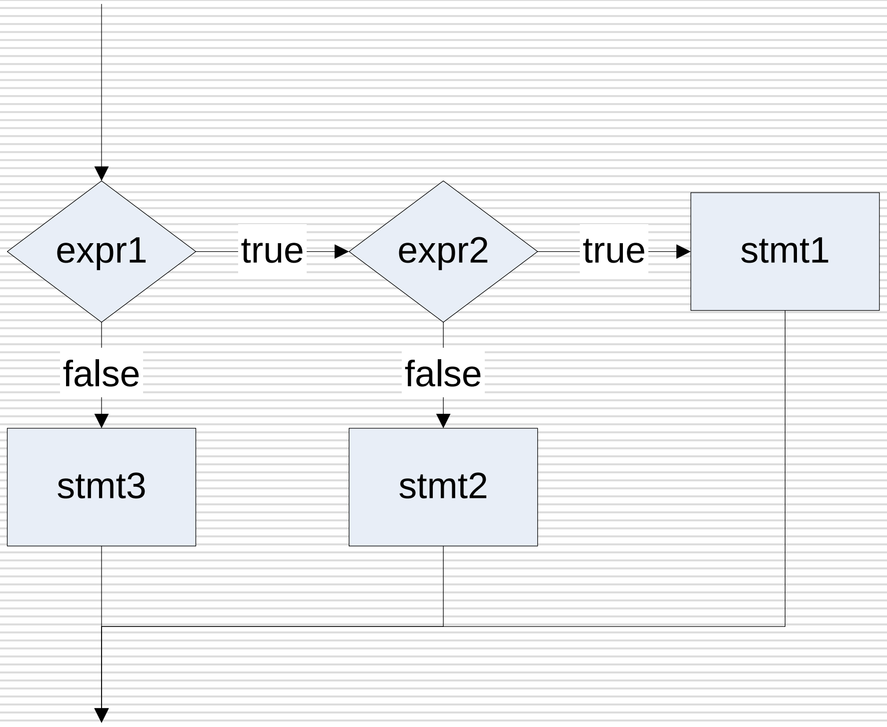
- if **expr** evaluates to nonzero (TRUE) then **stmt1** is executed
- if **expr** evaluates to zero (FALSE) then **stmt2** is executed



```
if (expr)
    stmt1;
else
    stmt2;
```

Nested if Statements

- when testing more than one condition

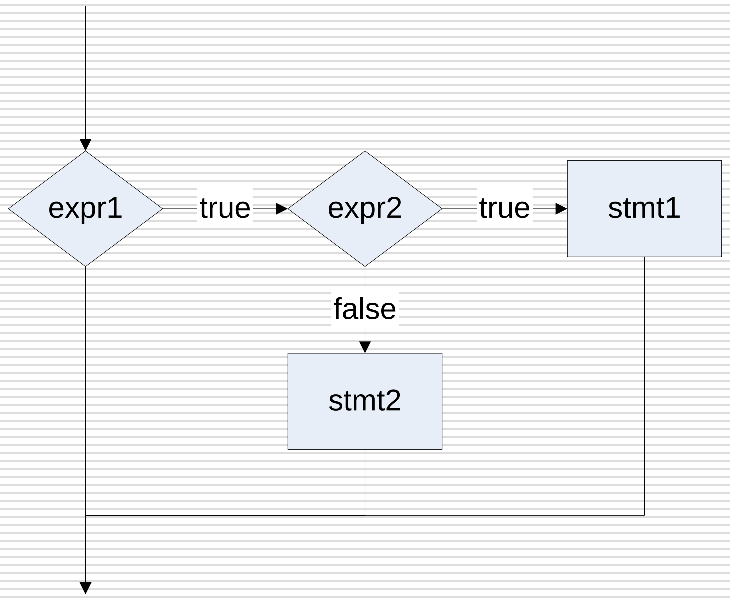


```
if (expr1)
    if (expr2)
        stmt1;
    else
        stmt2;
else
    stmt3;
```

***Note code indentation and alignment*.**

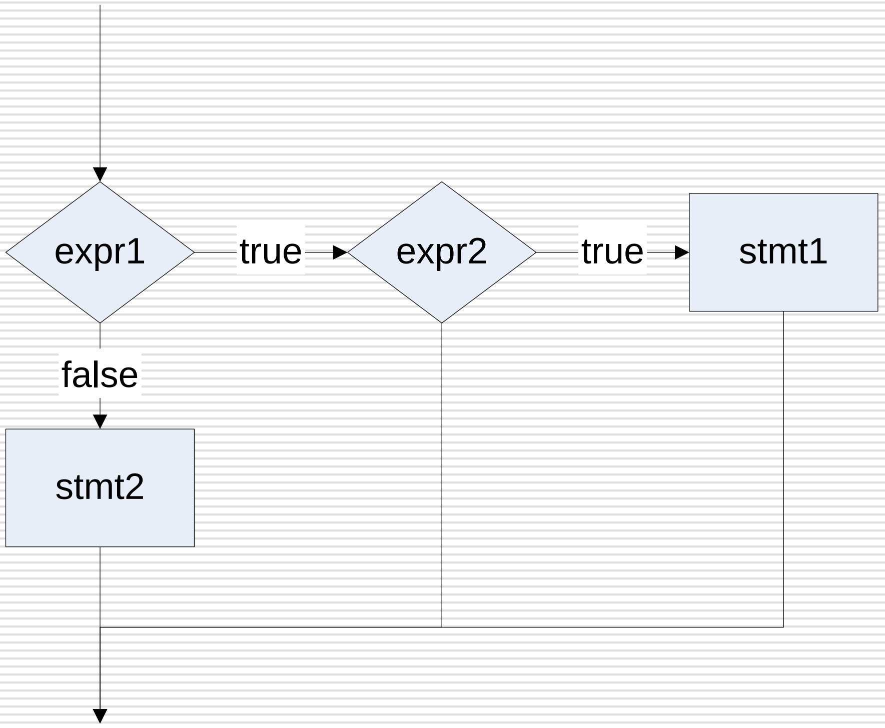
Nested if Statements

- when nested IF is not enclosed in braces, ELSE is associated with the inner IF statement



```
if (expr1)
  if (expr2)
    stmt1;
  else
    stmt2;
```

Nested if Statements



```
if (expr1)
{
  if (expr2)
    stmt1;
}
else
  stmt2;
```

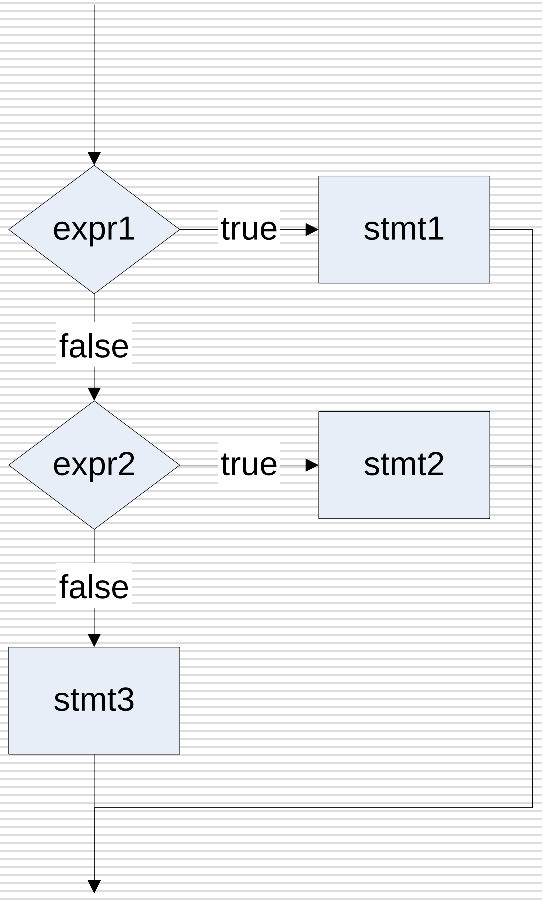
The if/else if Statement

- trailing ELSE provides default action when none of the expressions are TRUE

```
if (expr1)
  stmt1;
else
  if (expr2)
    stmt2;
  else
    stmt3;
```

OR

```
if (expr1)
  stmt1;
else if (expr2)
  stmt2;
else
  stmt3;
```



Flags

- Boolean or integer variable that signals when a condition exists

```
bool salesQuotaMet = false;

if (sales > QUOTA_AMOUNT)
  salesQuotaMet = true;

if (salesQuotaMet)
  cout << “You have met your sales quota!\n”;
--OR–
if (salesQuotaMet == true)
  cout << “You have met your sales quota!\n”;
```

Boolean flag

Flags

- Boolean or integer variable that signals when a condition exists

```
int salesQuotaMet = 0;

if (sales > QUOTA_AMOUNT)
  salesQuotaMet = 1;

if (salesQuotaMet)
  cout << “You have met your sales quota!\n”;
--OR–
if (salesQuotaMet == true)
  cout << “You have met your sales quota!\n”;
```

Integer flag

Logical Operators

- Connect two or more relational expressions into a single expression, or reverse logic of the expression

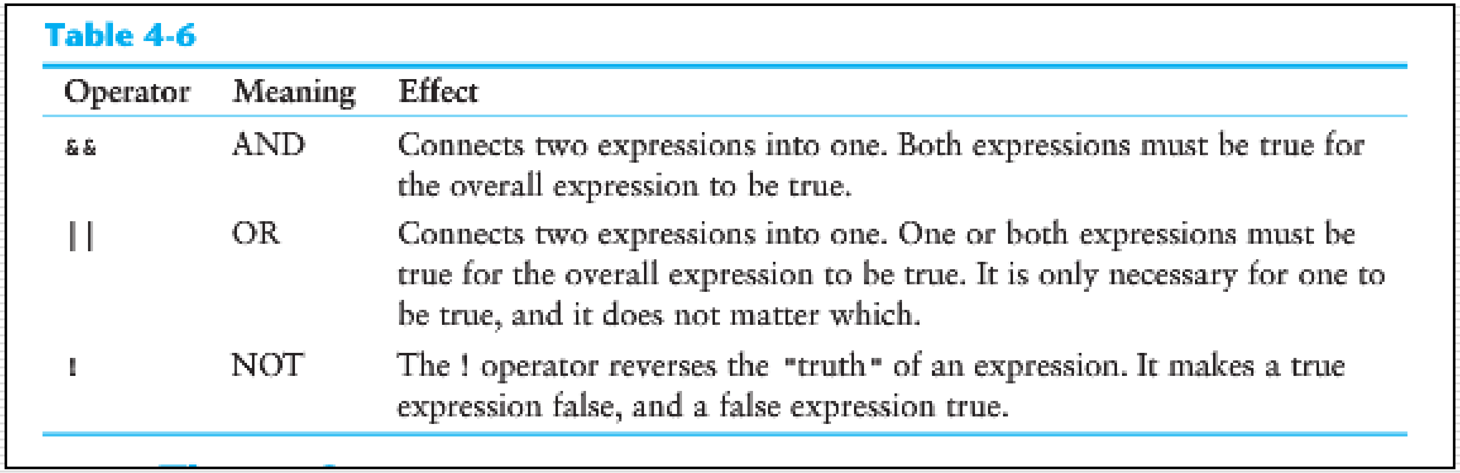


Table 4-6

Operator	Meaning	Effect
&&	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
||	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
!	NOT	The ! operator reverses the "truth" of an expression. It makes a true expression false, and a false expression true.

Logical Operators

- && logical AND
  - binary operator
  - evaluates TRUE only when both operands are TRUE

&&	T	F
T	T	F
F	F	F

```
if (temperature < 20 && minutes > 12)
  cout << “The temperature is in the danger zone.” << endl;

// range checking
if (x >= 20 && x <= 40)
  cout << x << “ is in the acceptable range.” << endl;
```

Logical Operators

- || logical OR
  - binary operator
  - evaluates TRUE when either operand is TRUE

||	T	F
T	T	T
F	T	F

```
if (temperature < 20 || temperature > 100)
  cout << “The temperature is in the danger zone.” << endl;

// range checking
if (x < 20 || x > 40)
  cout << x << “ is outside the acceptable range.” << endl;
```

Logical Operators

- evaluates to a TRUE/FALSE value after operand evaluation
- ! logical NOT
  - unary operator
  - forces operand to its opposite state

	!
T	F
F	T

```
if (!(temperature > 100))
  cout << “You are below the maximum temperature.” << endl;
```

Short Circuit Evaluation

- if result of compound expression can be determined before the entire expression is evaluated, remainder of the expression is not evaluated
  - improves run-time efficiency

&&	T	F
T	T	F
F	F	F

- &&
  - FALSE if first subexpression is FALSE
  - use to ensure objects have a particular property before being manipulated

```
(fVar >= 0) && (fVar = sqrt(iVar))
```

Short Circuit Evaluation

- if result of compound expression can be determined before the entire expression is evaluated, remainder of the expression is not evaluated
  - improves run-time efficiency

||	T	F
T	T	T
F	T	F

- ||
  - TRUE if first subexpression is TRUE
  - use to ensure objects have a particular property before being manipulated

```
    (fileExists) || (createFile())
```

Conditional Operator

- conditional operator works like if/else statement
  - ternary (three operands) operator
- format:
  - x1 ? x2 : x3;
    - if x1 is nonzero (TRUE), expression has data type and value x2
    - if x1 is zero (FALSE), expression has data type and value x3

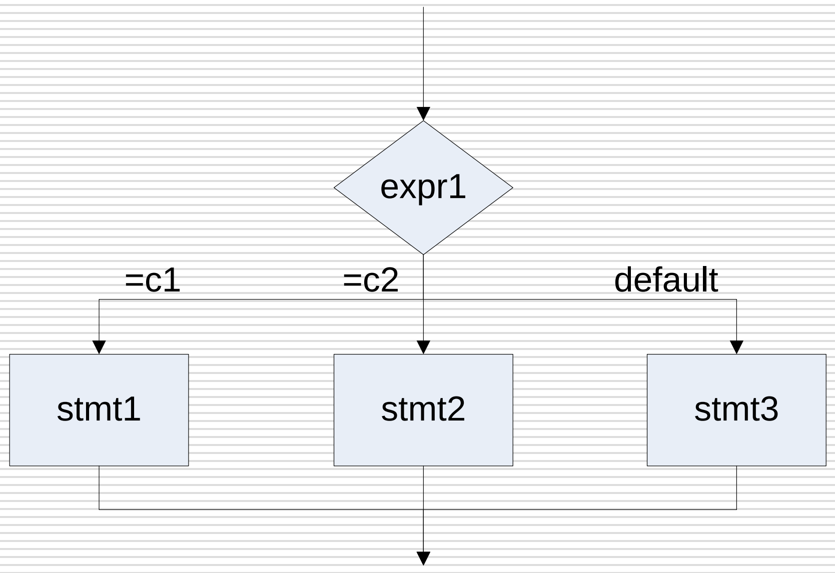
```
cout << “Your grade is: ”
  << (score < 60 ? “Fail.” : “Pass.”);
```

Conditional Operator

- conditional operator works like if/else statement
  - ternary (three operands) operator
- format:
  - x1 ? x2 : x3;
    - if x1 is nonzero (TRUE), expression has data type and value x2
    - if x1 is zero (FALSE), expression has data type and value x3

```
int Max(int a, int b)
{
    return a>b?a:b;
}
```

The switch Statement



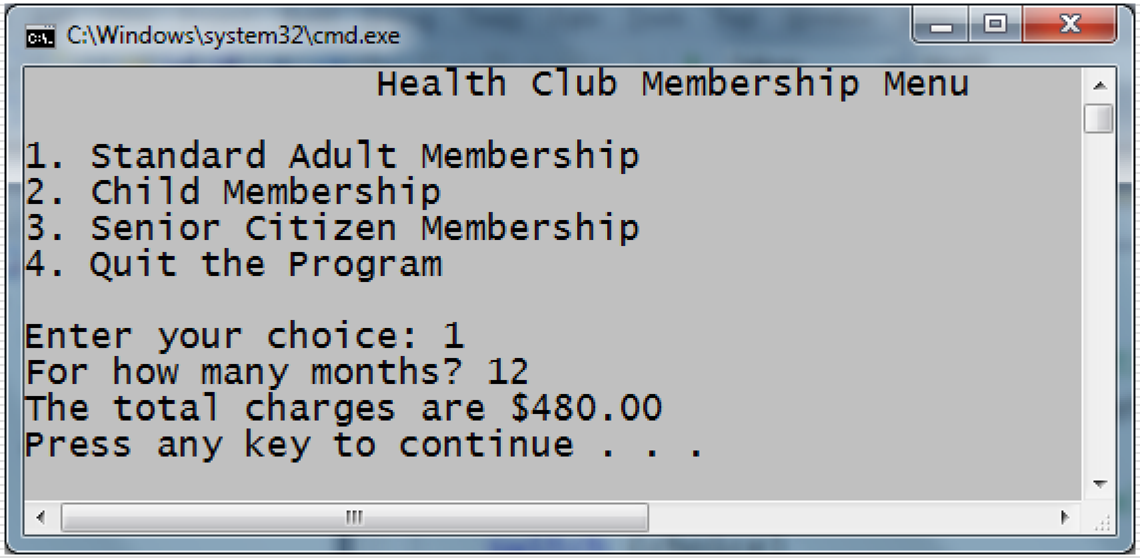
```
switch (expr)
{
    case c1:  stmt1;
              break;
    case c2:  stmt2;
              break;
    default:  stmt3;
}
```

The switch Statement

- **expr** is integral type
  - char, short, int, long, enum
- all constant expressions must be different
- only one default per switch
  - default is optional
- multi-way decision that branches accordingly and continues execution until a **break** statement or end of switch is encountered
  - no need to enclose multiple statements in braces { }

The switch Statement

- good for menu processing


C:\Windows\system32\cmd.exe

```
                Health Club Membership Menu

1. Standard Adult Membership
2. Child Membership
3. Senior Citizen Membership
4. Quit the Program

Enter your choice: 1
For how many months? 12
The total charges are $480.00
Press any key to continue . . .
```

The switch Statement

- good for menu processing

```
int choice;        // To hold a menu choice
int months;        // To hold the number of months
double charges;    // To hold the monthly charges

// Constants for membership rates
const double ADULT = 40.0,
             SENIOR = 30.0,
             CHILD = 20.0;
// Constants for menu choices
const int ADULT_CHOICE = 1,
          CHILD_CHOICE = 2,
          SENIOR_CHOICE = 3,
          QUIT_CHOICE = 4;

// Display the menu and get a choice.
cout << "\t\tHealth Club Membership Menu\n\n"
     << "1. Standard Adult Membership\n"
     << "2. Child Membership\n"
     << "3. Senior Citizen Membership\n"
     << "4. Quit the Program\n\n"
     << "Enter your choice: ";
cin >> choice;
```

The switch Statement

- good for menu processing

```
// Respond to the user's menu selection.
switch (choice)
{
   case ADULT_CHOICE:
      cout << "For how many months? ";
      cin >> months;
      charges = months * ADULT;
      cout << "The total charges are $" << charges << endl;
      break;
   case CHILD_CHOICE:
      . . .
      break;
   case SENIOR_CHOICE:
      . . .
      break;
   case QUIT_CHOICE:
      cout << "Program ending.\n";
      break;
   default:
      cout << "The valid choices are 1 through 4. Run the\n“
         << "program again and select one of those.\n";
}
```

Blocks and Variable Scope

- scope of a variable is limited to the block in which it is defined
  - after variable declaration

```
int main()
{
  int counter;
  cin >> counter;
  cout << counter << endl;
  return 0;
}
```

```
int main()
{
  cin >> counter;
  int counter;
  cout << counter << endl;
  return 0;
}
```

Blocks and Variable Scope

- scope of a variable is limited to the block in which it is defined
  - after variable declaration

```
int main()
{
  int counter;
  cin >> counter;
  cout << counter << endl;
  return 0;
}
```

```
int main()
{
  cin >> counter;
  int counter;
  cout << counter << endl;
  return 0;
}
```

Blocks and Variable Scope

- scope of a variable is limited to the block in which it is defined
  - after variable declaration

```
int main()
{
  bool isValid = true;
  if (isValid)
  {
      int counter;
      cin >> counter;
  }
  cout << counter << endl;
  return 0;
}
```

Blocks and Variable Scope

- scope of a variable is limited to the block in which it is defined
  - after variable declaration

```
int main()
{
  bool isValid = true;
  if (isValid)
  {
      int counter;
      cin >> counter;
  }
  cout << counter << endl;
  return 0;
}
```