

Example: Days of the Week

- Problem: convert from a number (1-7) to the name of the corresponding day of the week
 - Sunday is 1, Monday is 2, etc.
- Just like the months in the data conversion problem
 - Could use an if tree...

switch Structure

- Alternative to `if..else`
- Used with a finite set of values
 - Letter grades
 - Months of the year
 - Type codes
- `expression` is evaluated first (must be `integer`)
- Execution jumps to the corresponding `case`
- A `default` case may be included

```
switch (expression)
{
    case value1:
        statements1
        break;
    case value2:
        statements2
        break;
    .
    .
    .
    case valuen:
        statementsn
        break;
    default:
        statements
}
```

case, break and default

- Unlike `if...else`, each `case` in a `switch` is not a block of code
- `case` labels determine only where execution jumps to, not where it ends
- To skip the rest of the `cases`, you use `break`
 - (But you don't have to)

```
switch (expression)
{
    case value1:
        statements1
        break;
    case value2:
        statements2
        break;
    .
    .
    .
    case valuen:
        statementsn
        break;
    default:
        statements
}
```

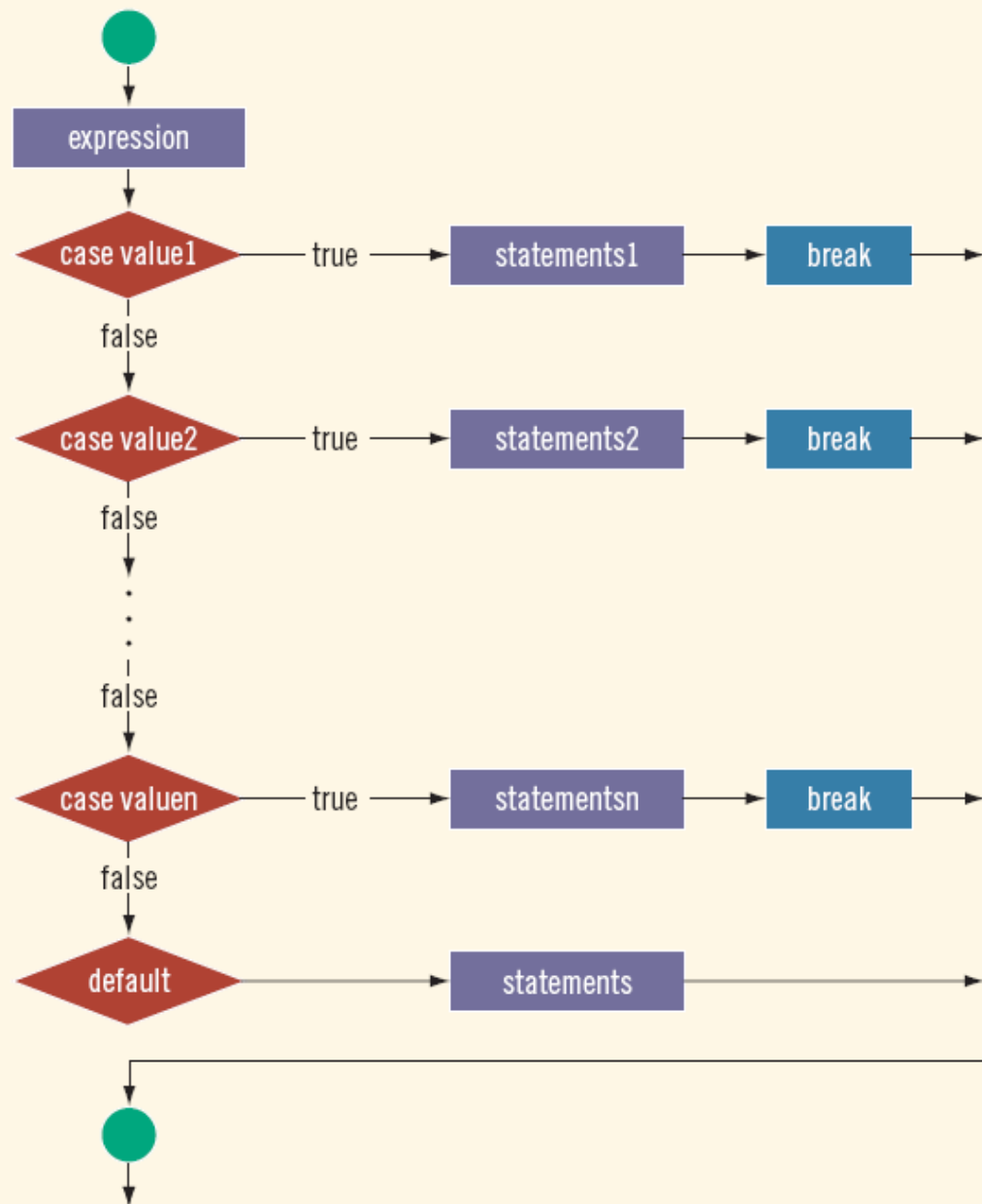


FIGURE 4-4 `switch` statement

EXAMPLE 4-24

Consider the following statements, where `grade` is a variable of type `char`:

```
switch (grade)
{
case 'A':
    cout << "The grade is 4.0.";
    break;
case 'B':
    cout << "The grade is 3.0.";
    break;
case 'C':
    cout << "The grade is 2.0.";
    break;
case 'D':
    cout << "The grade is 1.0.";
    break;
case 'F':
    cout << "The grade is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

In this example, the expression in the `switch` statement is a variable identifier. The variable `grade` is of type `char`, which is an integral type. The possible values of `grade` are 'A', 'B', 'C', 'D', and 'F'. Each `case` label specifies a different action to take, depending on the value of `grade`. If the value of `grade` is 'A', the output is:

The grade is 4.0.

Exercise

```
int unitID;  
double overheadRate;  
cin >> unitID;  
switch( unitID )  
{  
case 0:  
    overheadRate = 2.9;  
    break;  
case 1:  
case 2:  
    overheadRate = 3.4;  
    break;  
case 3:  
    overheadRate = 4.1;  
    break;  
default:  
    overheadRate = 5.0;  
}
```

What values for
this table
correspond to that
code?

Unit ID	Overhead Rate

More Interesting Version

```
int unitID;  
double overheadRate;  
...  
switch( unitID / 100 )  
{  
case 0:  
    overheadRate = 2.9;  
    break;  
case 1:  
case 2:  
    overheadRate = 3.4;  
    break;  
case 3:  
    overheadRate = 4.1;  
    break;  
default:  
    overheadRate = 5.0;  
}
```

What values for
this table
correspond to that
code?

Unit ID	Overhead Rate

Terminating a Program with the `assert` Function

- Certain types of errors that are very difficult to catch can occur in a program
 - Example: division by zero can be difficult to catch using any of the programming techniques examined so far
- The predefined function, `assert`, is useful in stopping program execution when certain elusive errors occur

The `assert` Function (continued)

- Syntax:

```
assert (expression) ;
```

- `expression` is any logical expression
 - If `expression` evaluates to `true`, the next statement executes
 - If `expression` evaluates to `false`, the program terminates and indicates where in the program the error occurred
- To use `assert`, include `cassert` header file

The `assert` Function (continued)

- `assert` is useful for enforcing programming constraints during program development
- After developing and testing a program, remove or disable `assert` statements
- To disable the `assert` statement:

```
#define NDEBUG  
#include <cassert>
```