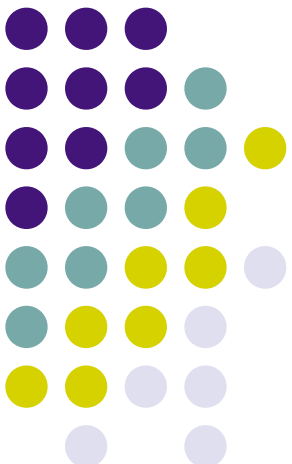
The C "switch" Statement

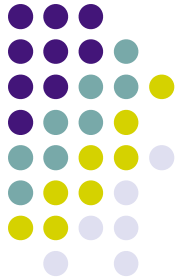
CMSC 104, Spring 2014

Christopher S. Marron

(thanks to John Park for slides)



The switch Statement



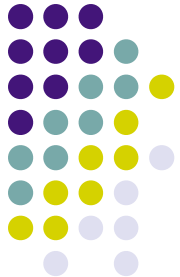
Topics

- Multiple Selection
- switch Statement
- char Data Type and getchar( )

Reading

- Section 4.7, 4.12

Multiple Selection

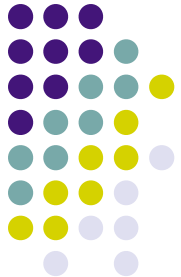


- So far, we have only seen **binary selection**.

```
if ( age >= 18 )
{
    printf("Vote!\n") ;
}
```

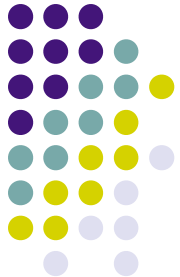
```
if ( age >= 18 )
{
    printf("Vote!\n") ;
}
else
{
    printf("Maybe next time!\n") ;
}
```

Multiple Selection (con't)



- Sometimes it is necessary to **branch** in more than two directions.
- We do this via **multiple selection**.
- The multiple selection mechanism in C is the **switch** statement.

Multiple Selection with if

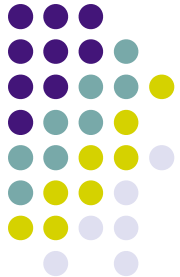


```
if (day == 0 ) {
    printf (“Sunday”) ;
}
if (day == 1 ) {
    printf (“Monday”) ;
}
if (day == 2) {
    printf (“Tuesday”) ;
}
if (day == 3) {
    printf (“Wednesday”) ;
}
```

(continued)

```
if (day == 4) {
    printf (“Thursday”) ;
}
if (day == 5) {
    printf (“Friday”) ;
}
if (day == 6) {
    printf (“Saturday”) ;
}
if ((day < 0) || (day > 6)) {
    printf(“Error - invalid day.\n”) ;
}
```

Multiple Selection with if-else

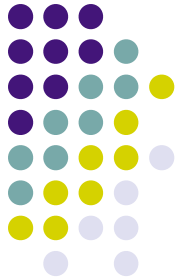


```
if (day == 0 ) {
    printf (“Sunday”) ;
} else if (day == 1 ) {
    printf (“Monday”) ;
} else if (day == 2) {
    printf (“Tuesday”) ;
} else if (day == 3) {
    printf (“Wednesday”) ;
} else if (day == 4) {
    printf (“Thursday”) ;
} else if (day == 5) {
    printf (“Friday”) ;
} else if (day == 6) {
    printf (“Saturday”) ;
} else {
    printf (“Error - invalid day.\n”) ;
}
```

This if-else structure is more efficient than the corresponding if structure. Why?

Are there any other functional differences?

Multiple Selection with if-else

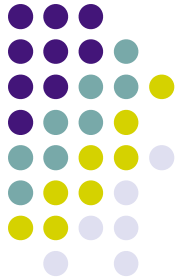


```
if (day == 0 ) {
    printf (“Sunday”) ;
    day = 3;
}
if (day == 1 ) {
    printf (“Monday”) ;
}
if (day == 2) {
    printf (“Tuesday”) ;
}
if (day == 3) {
    printf (“Wednesday”) ;
}
if (day == 4) {
    printf (“Thursday”) ;
}
…
```

VS.

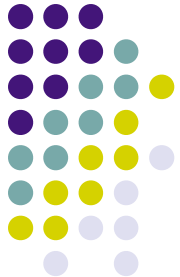
```
if (day == 0 ) {
    printf (“Sunday”) ;
    day = 3;
} else if (day == 1 ) {
    printf (“Monday”) ;
} else if (day == 2) {
    printf (“Tuesday”) ;
} else if (day == 3) {
    printf (“Wednesday”) ;
} else if (day == 4) {
    printf (“Thursday”) ;
} else if (…)
…
```

The switch Multiple-Selection Structure



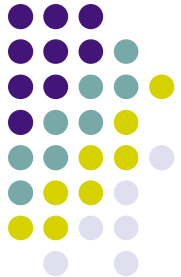
```
switch ( integer expression )
{
    case constant1 :
        statement(s)
        break ;
    case constant2 :
        statement(s)
        break ;
        . . .
    default:
        statement(s)
        break ;
}
```

switch Statement Details



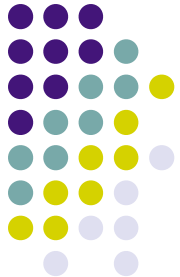
- The last statement of each case in the switch should almost always be a break.
- The break causes program control to jump to the closing brace of the switch structure.
- Without the break, the code flows into the next case. This is almost never what you want.
- A switch statement will compile without a default case, but always consider using one.

Good Programming Practices



- Include a default case to catch invalid data.
- Inform the user of the type of error that has occurred (e.g., “Error - invalid day.”).
- If appropriate, display the invalid value.
- If appropriate, terminate program execution (discussed in CMSC 201).

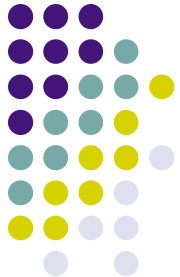
switch Example



```
switch ( day )
{
      case 0:  printf (“Sunday\n”) ;
               break ;
      case 1:  printf (“Monday\n”) ;
               break ;
      case 2:  printf (“Tuesday\n”) ;
               break ;
      case 3:  printf (“Wednesday\n”) ;
               break ;
      case 4:  printf (“Thursday\n”) ;
               break ;
      case 5:  printf (“Friday\n”) ;
               break ;
      case 6:  printf (“Saturday\n”) ;
               break ;
      default:  printf (“Error -- invalid day.\n”) ;
               break ;
```

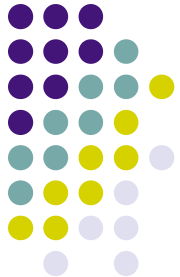
Is this structure more efficient than the equivalent nested if-else structure?

switch Example



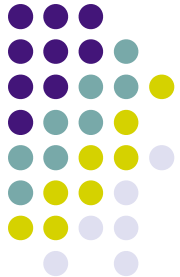
```
switch ( day )
{
      case 1:  printf (“Monday\n”) ;
               break ;
      case 2:  printf (“Tuesday\n”) ;
               break ;
      case 3:  printf (“Wednesday\n”) ;
               break ;
      case 4:  printf (“Thursday\n”) ;
               break ;
      case 5:  printf (“Friday\n”) ;
               break ;
      case 0:
      case 6:  printf (“Weekend\n”) ;
               break ;
      default:  printf (“Error -- invalid day.\n”) ;
               break ;
}
```

Why Use a switch Statement?



- A switch statement can be more efficient than an if-else.
- A switch statement may also be easier to read.
- Also, it is easier to add new cases to a switch statement than to a nested if-else structure.

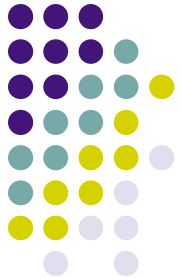
In-Class Exercise



Use nested loops to write a prime number calculator:

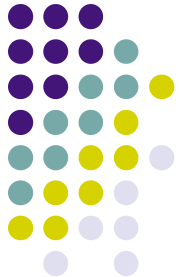
- Determine whether each member of a range of number is prime, by attempting to divide it evenly by each of the smaller numbers

In-Class Exercise



- General strategy:
- Prompt user for upper limit
  - Your program will then test all numbers from 2 to *limit*
- Outer loop: iterate over all numbers from 2 to *limit*, testing each in an inner loop to see if it's prime

In-Class Exercise



- Inner loop: You have the loop variable from the outer loop—let's say you called it *num_to_test*
- Iterate over all numbers from 2 to (*num_to_test* – 1) (why "- 1"?):
  - For each turn of the inner loop, test that number to see if it divides evenly into *num_to_test*
    - If it does, num_to_test is **not** prime!
- At end of inner loop, if you were never able to evenly divide num_to_test, it is prime—print that out to user