The “switch” statement and the “char” type

CMSC 104 Section 02
April 3, 2024

Administrative Notes

Homework 5 due tonight

Quiz 3 results

The grading scale (from syllabus)

How to code multiple choices

So far, we have only studied “if” and “if-else” statements

- “If” works great if there’s only one option - do this, or do nothing
- “If-else” works perfectly for two choices; do this, or do that
- If there are more than two choices we just nest “if-else” statements
  - As we have seen over the last few weeks, that can get very confusing
    - You have to figure out which “else” goes with which “if”
    - Usually by mapping { to }

Today, we’re going to learn an “easier” way to do it: the “switch” statement

We’ll start by working through a problem: if we encode the day of the week as an integer (0 means Sunday; 6 means Saturday), we want to print the actual day that corresponds to the integer

Solution 1: only “if” statements

```
if (day == 0) {
   printf("Sunday\n"); }
if (day == 1) {
   printf("Monday\n");}
if (day == 2) {
   printf("Tuesday\n");}
if (day == 3) {
   printf("Wednesday\n");}
if (day == 4) {
   printf("Thursday\n");}
if (day == 5) {
   printf("Friday\n");}
if (day == 6) {
   printf("Saturday\n");}
if (day > 6 || day < 0) {
   printf("Invalid day.\n");}
```

Solution 2: if-else statements

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

Solution 3: using the “switch” statement

```
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");
 }
```

Some notes on the “switch” statement

- The condition on which the switch chooses MUST be an integer or a char; NOT a boolean
  - If it’s some other type you have to figure out how to convert it to an integer, or you can’t use “switch”
- You define a case for each integer value of interest
- Cases must be unique; you can’t define cases that overlap. In other words, no value can match multiple cases
- Curly brackets {} around the statements executed for each case are optional. You do not need them. But it can be good to use them
- The “default” case is executed for all values for which no case is defined

What about those “break” statements?

Last week we learned that “break” was bad coding; and almost always the result of a ‘lazy’ coder who couldn’t think of the right way to structure a loop so that break wasn’t needed

- That’s true. But “switch” is different. It’s not a loop. “Break” means that we’re done with the switch statement, entirely
- What happens if you leave out a “break”?
- Control of the program flows through the rest of the cases in the switch statement
    - You may wind up executing multiple cases in the switch
    - Generally, you don’t want to do this.

Why Use a Switch Statement?

- A switch statement can be more efficient than an if-else.
- A switch statement may also be more readable.
- It is easier to add new cases to switch statement than to nested if-else statements

In short: you never have to use a switch statement in programming, but they can make programs easier to write/maintain/understand

The “char” datatype

The char data type holds a single character, or byte.

```
char ch;
```

Example assignments:

```
char grade, symbol;

grade = ‘B’;

symbol = ‘$’;
```

The char is held as a one-byte integer in memory.

The ASCII code is what is actually stored, so we can use the char type as characters or integers, depending on the requirements of the algorithm we are implementing.

The maximum value for a signed char is 127 (7 bits only zero to 127).

The maximum value for a unsigned char is 255 (8 bits/1 byte holds only zero to 255).

Input & Output with chars

Use

```
scanf("%c", &ch);
```

to read a single character into the variable ch, for example.

Use

```
printf("%c", ch);
```

to display the value of a character variable.

An example: scanf and char

```
#include <stdio.h>
int main() {
    char ch;
    printf("Enter a character: ");
    scanf("%c", &ch);
    printf("The value of %c is %d.\n", ch, ch);
    return 0; }
```

If the user would enter “A”, the output would be:

```
The value of A is 65.
```

An alternative to scanf: “getchar”

- The getchar() function is found in the stdio.h library.
  - So you can use it in your program without having to write any additional code
- The getchar() function reads *one character* from stdin (the standard input buffer) and returns that character’s ASCII value.
- The value can be stored in either a character or integer variable.

getchar() example

```
#include <stdio.h>
 int main() {
char ch;
printf("Enter a character: ");
ch = getchar(); /* same as scanf("%c", &ch); */
printf("The value of %c is %d.\n", ch, ch);
return 0; }
```

If the user would enter “A”, the output would be:

```
The value of A is 65.
```

Problems when reading characters

- When getting characters, whether using `scanf()` or `getchar()`, realize that you are reading only one character.
- What will the user actually type? The character followed by pressing ENTER.
- So, the user is actually entering two characters, their response & the newline character.
- Unless you handle this, the newline character will remain in the stdin stream causing problems the next time you want to read a character.
- You need another call to `scanf()` or `getchar()` to remove it.

Improved getchar() example

```
#include <stdio.h>
 int main() {
        char ch, newline;
        printf("Enter a character: ");
        ch = getchar(); /* same as scanf("%c", &ch); */
        scanf("%c", &newline); /* grab the newline character */
        printf("The value of %c is %d.\n", ch, ch);
        printf("Enter another character: ");
        scanf("%c", &ch);
        scanf("%c", &newline);
        printf("The value of %c is %d.\n", ch, ch);
return 0; }
```

Additional concerns when reading in data

- When we were reading integers using scanf(), we didn’t seem to have problems with the newline character, even though the user was typing ENTER after the integer.
- That is because scanf() was looking for the next integer and ignored the newline (whitespace).
- If we use scanf("%d", &num); to get an integer, the newline is still stuck in the input stream.
- If the next item we want to get is a character, whether we use scanf() or getchar(), we will get the newline.
- We have to take this into account and remove it