CS2303 In Class Exercises

Jan 17, 2020

January 9, 2020

Abstract

Today's goal is that you can write functions using statements in C.

Name and WPI email: Kush Shah kshah2@wpi.edu

1 Statements in C

Our first division of the statements in C is between those we recommend using, and those we discourage.

1.1 Recommended Statements

1.1.1 expression statement

This includes the assignment, and the function invocation, both very frequently used forms, for example:
x = 5;
x++;
function(<arguments>);

1.1.2 if, if/else statement

This statement allows the expression of a condition. The condition is evaluated into true or false. If true, the block of statements following the condition is executed. Otherwise that block is skipped. If there is an else clause, the block of statements in the else clause is executed when the condition evaluates to false.

Example:

```
if(tests())
{
    puts("About to run production.");
    production(argc, argv);
}
```

```
else
{
    puts("Tests did not pass.");
}
```

Note in the example above that the condition evaluates to a logical (bool), and can be a function invocation. The example uses the optional else clause. Though, in the event only one statement makes up the if clause body or the else clause body, the curly braces are optional, it is a popular programmer error to add another statement to such a body while forgetting to add the curly braces. Develop the habit of always using curly braces in the if and the else clause, to protect yourself from this common error.

Exercise:
Code an if statement that checks whether a variable x is equal to 3, and increments x if it is, and substracts one from x if it is not.

```
if(x==3){
x++;
}else{
x--;}
```

1.1.3 switch/case statement

The flow of control achieved with the switch/case construct can also be achieved with nested if/else statements. The switch/case statement can be easier to read. An expression is evaluated, and depending upon its value, a case is chosen. The code block associated with that case is executed. This context is the only context, in terms of good programming practice, in which to use break. Upon encountering a break, flow of control moves to after the switch case construct. This may involve executing though case labels. The last case of a switch case should only be used for unanticipated events.

Example:

```
switch(i)
{
  case 1:
    //this is filename
        printf("The length of the filename is %d.\n",strlen(argv[i]));
        printf("The proposed filename is %s.\n", argv[i]);
        if(strlen(argv[i])>=FILENAMELENGTHALLOWANCE)
        {
            puts("Filename is too long.");fflush(stdout);
            answer = false;
        }
        else
        {
             strcpy(filename, argv[i]);
             printf("Filename was %s.\n", filename);fflush(stdout);
        }
        break;
   case 2:
     //this is maximum number of rooms

       aL= strtol(argv[i], &eptr, 10);
       maxRooms = (int) aL;
       printf("Number of rooms is %d\n",maxRooms);fflush(stdout);
       break;
   case 3:
     //this is maximum amount of treasure

       aL= strtol(argv[i], &eptr, 10);
       maxTreasure = (int) aL;
       printf("Amount of treasure is %d\n",maxTreasure);fflush(stdout);
       break;

   default:
      puts("Unexpected argument count."); fflush(stdout);
      answer = false;
      break;
 }//end of switch
```

Note in the example above that the switch value evaluates to an integer that can be matched against the values shown in the case labels.
Because in this example, every case body ends with a break statement, prior to the next case label, we do not see execution of multiple case bodies. By contrast, if a break statement does not appear, execution will continue undeterred by a case label.

Enumerated types are compatible with the needs of the switch/case statement.

Exercise: (As always, if you have questions, ask. After this occasion, this material becomes background knowledge.)
Write an enumerated type for day of the week. It is ok to omit weekend days. Declare and initialize a variable of your datatype. Write a switch/case statement that has case labels for each day of your week.

```
typedef enum{
 Monday,
 Tuesday,
 Wednesday,
 Thursday,
 Friday} days;
days day = Monday;
switch(day){
case Monday:
printf("monday");
break;
case Tuesday
printf("tuesday");
break;
case Wednesday:
printf("wednesday");
break;
case Thursday:
printf("thursday");
break;
case Friday:
printf("friday");
break;
default:
printf("wrong");
break;}
```

1.1.4 for statement

The for statement provides three components for specifying repetition: Initialization, termination and changes after a repetition. These control repetition such that, before the first repetition, the initialization is carried out. Likewise, the termination condition is checked. If the termination condition is not yet met, the block of code governed by the repetition conditions is executed. After the block is executed, the termination condition is checked. If the termination condition is not yet met, the changes to be performed after a repetition are carried out. Then the block of code governed by the repetition conditions is executed again.

Example:

```
for(int i = 1; i<argc; i++) //don't want to read argv[0]
{//argv[i] is a string

...

}
```

In the example above, we observe that the initialization, termination and changes after a repetition are separated from each other by the two semicolons. Though the example initializes only one variable, it is possible to initialize more than one variable. When multiple initializations are desired, they are separated by commas. The termination condition appears next. It must evaluate to true or false. It is possible that several logicals are combined into a logical. We can use logical operators such as && or || to perform logical operations, in the termination condition component. The last of the three parts is the changes to be performed after an execution of the body. As with the initialization, there may be several such updates. Each is separated from the other by a comma.

Exercises: (As always, if you have questions, ask. After this occasion, this material becomes background knowledge.)

1. Write a for statement that initializes two variables, for example i and j, and adds one to i, and 2 to j, after an iteration, and terminates when j¿5.

   for(i = 0, j =0; j<=5;i++,j+=2){...}

2. Write a for statement that makes no update.

   for(int i = 0; i<100;){...}

1.1.5 while statement

The while statement, by contrast with the for statement, has a termination condition. You can think of it as a for statement with no initialization and no changes to be made after an execution of the body. This condition is checked prior to executing the body. Note that if the condition is initially false, the body will not be executed at all. On the other hand, if the condition is true, the body will be executed as many times as it takes for the condition to become true. A corollary is, it is mandatory that the condition change from true to false at some point in the execution, otherwise, the repetitions of the body will never cease. This popular programming error is called "infinite loop".

Example:

```
LLNode* temp = lp;
while(temp->next)
{
   temp=(LLNode*)temp->next;
}
//now temp points to the last element
```

In the example above we see that a variable, temp, of datatype pointer to (address of) some user-defined datatype LLNode is initialized to a value. Then the pointer temp is used to access of field of the datatype, called next. As long as the next field is not zero, the variable temp is updated to point to (hold the address of) the next field. LLNode is a user-defined datatype for an element of a linked list. Linked list elements contain an address of the next element. If there is no next element (as for the last element in the list) then the next field is set to zero.

Exercises: (As always, if you have questions, ask. After this occasion, this material becomes background knowledge.)

1. Write a while statement that tests one variable, called done, to see whether it is false.

   while(!done){...}

2. Write a while statement that tests whether both of two variables are true.

   while(x&&y){...}

3. Write a while statement that tests whether at least one of two variables is true.

```
while(x||y){..}
```

1.1.6 do-while statement

The do-while statement executes the body once before testing whether the body should be executed again. Thus the do-wile is very similar to a while. Imagine that a while statement is there, but the process of testing is skipped the first time.

Example:

```
int i = 0;
do
{
    printf("i is %d.\n",i);
} while (i<0);
```

Note that the example above will print “i is 0.” to the console. Exercises: (As always, if you have questions, ask. After this occasion, this material becomes background knowledge.)

1. Write a do-while loop that will execute the body three times.

```
int i = 0;
do{
body()
i++;
}while(i<3);
```

2. Imagine invoking a function. Write code, containing a while loop, that will invoke the function as if a do-while loop had been used.

1.1.7 return statement

This statement should occur at most once per function. Void functions do not use this statement. Non-void functions should use this statement once, at the bottom of the function. Other means should be used to control flow such that the function returns at the bottom. The reasons for this come from research into programmer errors especially in maintenance.

Example:

```
bool tests()
{
    bool answer = true; //so far
    bool ok1 = testReadFile();
    bool ok2 = testGotAdjacencyMatrix();
    bool ok3 = testSomethingElse();
    bool ok4 = testRemoveFromList();
    answer = ok1 && ok2 && ok3 && ok4;
    return answer;
}
```

Note in the example above that as soon as any of the variables whose names start with "ok" is false, the answer is known. Nevertheless, each test function is invoked, and the return value is calculated and returned at the end.

Recursive code is a recommended style, and encourages the use of recursive data structures.

Example:

```
bool charlie(int coins)
{
    bool done = false;
    if(coins<=0)
    {
        done = true;
    }
    if (!done)
    {
        puts("call charlie");
        done = charlie(coins-1);
    }
    return done;
}
```

Note in the example above, it is not necessary to return upon detecting the termination condition of the recursion.

1.2 Discouraged Statements

1.2.1 break

Only use this in the switch/case construct.
There are better ways to accomplish any of purpose you might envision for the break statement.

1.2.2 continue

Don't use this statement.
There are better ways to accomplish any of purpose you might envision for the continue statement.

1.2.3 goto and labeled statements

Don't use these.
There are better ways to accomplish any of purpose you might envision for these statements.

1.3 GOTO Considered Harmful

A much respected computer scientist, Edsger (yes, this is spelled correctly) Dijkstra, discouraged use of statements of the class "goto". His note can be read at `https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf`. Since 1968 further research on maintainability of software has supported this discouragement. Software managers and others who might interview you for jobs are aware of this, so please avoid use of discouraged statements.

Exercises: (As always, if you have questions, ask. After this occasion, this material becomes background knowledge.)

1. Write a function definition that has one bool parameter, returns a bool, and uses an if statement in the function body.

```
bool func(bool para){
    if(para){...}}
```

2. Write the function prototype for your function.

```
bool func(bool);
```

3. Give an example invocation of your function.

```
func(myBool);
```

4. Modify your function to include a for loop. Provide your code. Does the prototype or invocation change?

```
bool func(bool para){
  for(int i = 0; i< 2; i++){
    if(para){...}}}
```

the prototype and invocation do not change

5. Write a function that has one integer parameter, returns a double, and uses a switch/case statement in the function body.

```
double numFunc(int x){
bool even;
switch(x):
     case x%2==0:
      even = true;
       break;
     case x%2!=0:
      even = false;
      break;
    default:
      printf("incorrect type");
      break;
if(even){
return 1.0;}else{
return 2.0;}}
```