From Pseudcode Algorithms directly to C++ programs

(Chapter 7)

Part 1: Mapping Pseudo-code style to C++ style

input, output, simple computation,
lists, while loops, if statements

a bit of grammar

Part 2: Details
compilers, parsing, data types

How does learning a high level computer language add to understanding computation?

- Connects conceptual model of computation (finite state machine) with an actual physical machine (digital computer)
  - *We return to the physical elements of the machine in 2nd half of the course*
- Our finite state machine work connects the need for a precise grammar that indicates how to construct sentences that the machine can map to state transitions it can execute.
- The 'rules' that must be followed in communicating an algorithm to a machine using a high level language reflect
  - certain realities of how a physical machine represents & manipulates symbols that mean whatever we want them to mean

    3 "dog" *equal to* +
  - how computation is a series of state transitions

Consider assignment 1's finite state machine problems

Legal sentences are ones like

the large brown fox runs across the large field near the blue bridge
the clever bridge runs over the blue field

Illegal sentences are ones like

fox across brown

an fox runs across the field

near red fox runs

a cat runs across the field

a large brown runs large field the across fox

article = {a | the}
adj = {large | brown | red | blue }
noun = {fox | bridge | dog | field}
preposition = {near | across }

The main portion of a computer program consists of

1.specifying *what* goes in the machine's state (start state)

all the variables that will be manipulated

2. getting values *into* the machine state start state

3. specifying further instructions that set variable values, compare them, change them

4. getting values *out* of the machine state (e.g., the answer, the solution)

Mapping Pseudo-code to C++ code – simple input, computation, and output

What we said in Pseudo code	*Corresponding C++ Instruction*
Print the message "Enter 2 numbers"	**cout <<** "Enter 2 numbers ";
Get a value for the variable X	**cin >>** X;
Get a value for the variable Y	**cin >>** Y;
Set W to X - Y	W = X - Y;
Print the value of the variable W	**cout <<** W;

Pseudo code	the body of a C++ program
	{
specify the contents of the machine state—instructions reference these things	**int** limit, floor, max;
Print the message "Enter 2 numbers"	**cout <<** "Enter 2 numbers";
Get a value for the variable limit	**cin >>** limit;
Get a value for the variable floor	**cin >>** floor;
Set max to limit + floor	max = limit + floor
Print the value of the variable limit	**cout <<** max;
	}

```
{     int limit, floor, max;

      cout << "Enter 2 numbers";
      cin >> limit;
      cin >> floor;
      max = limit + floor;
      cout << max;
}
```

Not legal: max is not defined

```
{     int  limit, floor;

      cout << "Enter 2 numbers";
      cin >> limit;
      cin >> floor;
      max = limit + floor;
      cout << max;
}
```

Legal, but strange, for illustration…

```
{
int  limit, floor, max, dog, x, counter;

 cout << "Enter 2 numbers";
 cin >> limit;
 cin >> floor;
 max = limit + floor;
 cout << max;
}
```

recall Assignment 1: noun = {fox | bridge | dog | ….}

Simple vs. Structured Variables

Pseudocode *Get value of a variable called length*

```
cin >> length;
```

Pseudocode *Get a list of scores* $score_1, score_2, score_3, score_4$

```
cin >> score[1];
cin >> score[2];
cin >> score[3];
cin >> score[4];
```

Pseudocode *Set total to the sum of the scores* $_{1..4}$,

```
total = scores[1] + scores[2] + scores[3] + scores[4];
```

---

pseudocode *print out all the values of* $scores_{1..4}$ *and total*

```
cout << score[1];
cout << score[2];
cout << score[3];
cout << score[4];
cout << total;
```

Lists in C++ are called arrays

```
{       int  length, total, scores[4];

        length = 4;

        cout << "Enter 4 scores";

        cin >> scores[1];
        cin >> scores[2];
        cin >> scores[3];
        cin >> scores[4];

        total = scores[1] + scores[2] + scores[3] + scores[4];

        cout << scores[4], scores[3], scores[2], scores[1];
        cout << total;

}
```

Not legal:

```
cin >> scores;
cin >> scores[5];
cout << scores;
```


Write a program that asks for 1000 integers, puts them in a list, totals them, averages them, sorts them, whatever…

```
{       int  length, total, scores[1000];

        length = 1000;

        cout << "Enter 1000 scores";

        cin >> scores[1];
        cin >> scores[2];
        cin >> scores[3];
        cin >> scores[4];
        cin >> scores[5];
        cin >> scores [6];
        ….                                  // all the other cin's have to be written in
        cin >> scores[1000];

        total = scores[1] + scores[2] + scores[3] + scores[4] + scores[5] +  {every single one!}

}
```

loops allow the repetition of the same basic instructions….

```
int  length, total, scores[1000], counter;

length = 1000;

cout << "Enter 1000 scores";

counter = 1;                                  // why ?? //
while (counter <= length)

      {       cin >> scores[counter];

              counter = counter + 1;          // why?  //
      }

counter = 1;                                  // why ?? //
total = 0;                                    // why ?? //

while (counter <= length)
      {
            total = total + scores[counter];
            counter = counter + 1;
      }
```


Write the C++ program body that will
ask the user to enter a maximum and minimum value
ask the user to enter 1000 scores
put them in an array (a list)
As it puts them in the list, keeps track of how many scores are above the max value and below the max value

---

Pseudocode:

get the values for max, min, and $scores_1$.. $scores_{1000}$

set counter to 1
set length to 1000

set totalLow to 0 *a variable to keep track of how many below the min*
set totalHigh to 0 *a variable to keep track of how many above the max*

while (counter <= length)
get a value for $scores_{counter}$

if ($scores_{counter}$ < min) then
set totalLow to totalLow +1
else if ($scores_{counter}$ > max) then
set totalHigh to totalHigh + 1
set counter to counter + 1

C++ program body

```
int  max, min, totalLow, totalHigh, counter,  scores[1000];

cout << "Enter the high value, and then the low value ";
cin >> max;
cin >> min;

counter = 1;
length = 1000;

totalLow = 0;
totalHigh = 0;

while (counter <= length)
       {
         cin >> scores[counter];

         if  (scores[counter] < min)       // no then!
                    totalLow = totalLow +1;

         else if (scores[counter] > max)
                   totalHigh = totalHigh + 1;

         counter = counter + 1;
       }                                              // end of the while loop
```


Syntax for if-statements

Simple:

```
        cin >> item;
        cin >> min;

        count = 0;

        if (item < min)  count = count + 1;

        cout << "done";
```

Multi way Branching

```
        if (item < min)

                lowcount = lowcount + 1;

        else if (item > max)

                highcount = highcount + 1;

        cout << "done;"
```

Compound if statement must use { }

```
if (item < min)
    {
        lowcount = lowcount + 1;
        cout << "found a low value";
    }

cout << "all done";
```

Compare:

```
if (item < min)
    lowcount = lowcount +1;
    cout << "found a low value";
cout << "all done";
```

What the software checking your program operates on (recall FSM examples..}

```
if (item < min) {lowcount = lowcount + 1;cout << "found a low value";} cout << "all done";

if (item < min)  lowcount = lowcount +1;cout << "found a low value"; cout << "all done";
```


SUMMARY-1

- **There is a fairly straightforward mapping from pseudo-code to high level statements in C++**
- **punctuation counts. Indentation does not, except for humans**
- **C++ has a grammar—the instructions given for the program are being validated against the grammar**
- **the program body consists of**

DECLARATIONS	***what variables constitute the state***
legal C++ instructions	***the state transitions*** **get values in, set values, compare values, branch, loop, output values, etc.**