CSI 32
Chapter 4: Computation

Topics to discuss:

- Computations
- Objectives and tools
- Expressions
- Selection / Decision statements
- Iteration / Looping statements

Computation

- From one point of view, all that program ever does is to compute:
  - take some input
  - produce some output
- **Input**: keyboard, mouse, file, touch screen, etc
- **Output**: screen, file, network connection, other programs, other parts of program, etc.

Computation

- From one point of view, all that program ever does is to compute:
  - take some input
  - produce some output
- From a programming point of view the most important and interesting categories of input/output are “to/from another program” and “to/from other parts of a program”

- **Input**: keyboard, mouse, file, touch screen, etc
- **Output**: screen, file, network connection, other programs, other parts of program, etc.

Computation

- From one point of view, all that program ever does is to compute:
  - take some input
  - produce some output
- From a programming point of view the most important and interesting categories of input/output are “to/from another program” and “to/from other parts of a program”
- **Input**: keyboard, mouse, file, touch screen, etc
- **Output**: screen, file, network connection, other programs, other parts of program, etc.
- Most of the rest of the book could be seen as discussing “how do we express a program as a set of cooperating parts and how can they share/exchange data”?

Computation

- Computation is an act of producing some outputs based on some inputs.

Objectives and tools

- Our job as programmers is to express computations
  - correctly
  - simply
  - Effectively
- These concerns become ours the minute we start writing the code for others and accept that responsibility to do that well.

Objectives and tools

- The main tool for organizing a program (and our thoughts as we program) is to *break up a big computation into many little ones*.
  - using *abstraction*, i.e. hide details we don't need to use a facility,

    for example: use built-in procedure `sort` to sort a phone book
  - using *divide and conquer* technique, i.e. take a large program and divide it into several little ones

    for example: when building a dictionary, separate the job into three:
    - read the data
    - sort the data
    - output the data

Objectives and tools

- Why does “organizing the program” help?
  - We are not good at dealing with large problems
  - A 1000-line program is likely to have far more than 10 times as many errors as a 100-line program
  - A program built out of parts is likely I be slightly larger than a program where everything is optimally merged together
  - We cannot write and maintain large monolithic programs, so for large program with , say 10,000,000 lines, applying *abstraction* and *divide-and-conquer* is not just an option, but is an essential requirement

Expressions

```
int a = 3, b = 4, c = 5;
a + b
return a*a + b*b == c*c
```

- *Expression* is the most basic building block of programs
- 3, 4, and 5 are literals to initialize the variables a, b, and c (simplest expressions)
- Names of variables are also expressions
- Assignment expressions
- Expressions with comparison

Constant Expressions

```
const double p = 3.14;

constexpr double p2 = 3.14;


p = 3.141592654; // error
p2 = 3.1416;     // error
p + 1  // a constant
       // expression
```

- Programs typically use a lot of constants
- We want meaningful names for those constants
- C++ 98 didn't have `constexpr`, so people used `const`

Operators

- Most of C++ operators are similar to Python operators (see pages 97-98)
- ++val prefix increment / pre-increment
- val++ postfix increment / post-increment
- --val prefix decrement / pre-decrement
- val-- postfix decrement / post-increment

  see the program incrementExamples.cpp
- Word of caution: a < b < c doesn't evaluate as "b is between a and c"
  - a < b < c means (a<b)<c, and since (a<b) evaluates to a Boolean value, we will end up with either true < c or false < c.

    true is equivalent to 1, and false is equivalent to 0 see example.cpp

Statements

An *expression statement* is an expression followed by a semicolon

```
p = x; // assignment statement
p++;   // expression statement, increment statement
cout << p;   // output statement
```

In general, we want a statement to have some effect

```
if (x==5);   // an empty statement, doing nothing
{y=2;}       // y gets the value of 2
```

Selection / Decision statements

`if`-statements are a kind of selection / decision process

```
if (x==5)
    cout << “x is equal to 5\n”;
else
    cout << “x is equal to 5\n”;
```

Selection / Decision statements

`if`-statements are a kind of selection / decision process

```
if (x==5)
    {
      cout << “x is equal to 5\n”;
      return 5;
    }
else {
    cout << “x is equal to 5\n”;
    return -1;}
```

Selection / Decision statements

`if- else if` - else statements, no `elif` like in Python

```
if (x > 5)
    cout << “x is less than 5\n”;
else if (x > 5)
    cout << “x is greater than 5\n”;
else
    cout << “x is equal to 5\n”;
```

Selection / Decision statements

`switch`-statements are another kind of selection / decision process

```
switch(x){
    case ‘a’:
        cout << “letter a\n”;
        break;
    case ‘b’:
        cout << “letter b\n”;
        break;
    default:
        cout << “not a nor b\n”;
}
```

the value of x is compared against many constants.

see switchExample1.cpp and switchExample1.cpp

while-statements (iteration)

`while`-statements are similar to Python statements, with conditionals enclosed into parentheses

```
int i = 0;

while (i < 100){
    cout << i << “squared is ” << i*i << endl;
    ++i;
}
```

A sequence of statements enclosed by curly braces is called a *block* or a *compound statement*.

Empty block { } can be used to show that nothing is to be done.

for-statements (iteration)

`for`-statements are different from for statement in Python.

```
for(int i=0; i < 100; i++)
    cout << i << “squared is ” << i*i << endl;
```

while and for loops

```
int i = 0;

while (i < 100){
    cout << i << ": "
         << i*i << endl;
    ++i;
}
```

```
for(int i=0; i < 100; i++)
{
    cout << i << ": "
         << i*i << endl;
}
```

while and for loops

```
int value, s = 0;

while (cin >> value)
    s + = value;
cout << “their sum is “
    << s;
```

```
int s = 0;

for(int value; cin >> value;)
    s += value;
cout << “their sum is “
    << s;
```

To-do: grab any of the code fragments (while or for loop).
Change them to accept decimal values (use double type).
Change the loop to find the sum of the squares of entered values.
Run the program.
Enter 5 – 10 decimal values (separated by space of by an Enter), then press Ctr-Z and hit Enter.