COMP 102: Computers and Computing

Lecture 5: What is Programming?

Instructor: Kaleem Siddiqi (siddiqi@cim.mcgill.ca)

Class web page: www.cim.mcgill.ca/~siddiqi/102.html

Motivation

- The advantage of a computer it its ability to solve almost any problem.
- **BUT**: You need to explain what you want!

- With the right programming, a computer can:
  - Create movie animations.
  - Compose music.
  - Play chess.
  - At a low-level, all programs are implemented with Boolean logic blocks and finite-state machines.

- Programming gives you a higher-level way of expressing problems.
- It also means you can think about problems at a higher-level.

A programming language

- A programming language is a set of building blocks for constructing computer software.

- Like human language, there is a vocabulary and a grammar.
  - Sometimes called “semantic” and “syntactic” rules.

- Unlike human language, there is a very precise meaning for every word and sentence.
  - This is necessary so that the computer knows exactly how to interpret your instructions.

Languages

- Many different languages have been devised for people to communicate.
  - How many languages do you speak?
- Many different languages have been devised to communicate with computers.
  - How many can you name?
- Why do we need many languages?

Languages

- Each language provides a way of writing a kind of script along with rules for the computer to interpret the script as instructions.

- They can do very similar things, but some things are easier to say in some languages than in others.

Programming languages

- Tens of different programming languages have been created.
  - E.g. Java, C, C++, Perl, Python, O'Caml, Lisp,…
- Each has its own vocabulary.
  - Elements of the vocabulary can be combined to define new concepts.
- Each has its own syntax.
- Each language is good at describing different things.

A computer program

- Simplest definition of a program: "A sequence of instructions that a computer can interpret and execute".

- Is the following a computer program?

  "Analyze the US Census!"

- No! As stated, it cannot be understood and therefore cannot be executed by the computer. It is an instruction written in English.

- With computers, it is important to be very precise!
  - What is the census, what specific questions, with what precision, ...

Recall 4 key steps of computing

- Input
- Output
- Calculate
- Memorize

Programming languages need to handle all these components.

A computer program

- In programming, need to deal with 2 kinds of things:
  - Data:
    - Input
    - Output
    - Intermediate
  - Procedures:
    - List of instructions.
    - Each instruction tells the computer to do something.
    - Instructions are ordered correctly.

A Recipe for Scrambled Eggs

- Ingredients:

  2 eggs, 1 tbsp oil, salt

- Instructions:

  Step 1: Add oil to pan.
  Step 2: Heat pan on stove.
  Step 3: Crack 1st egg into pan.
  Step 4: Crack 2nd egg into pan.
  Step 5: Add salt.
  Step 6: Mix until light and cooked.

- Output:

  Scrambled eggs!

A recipe is a series of steps.

What if we did not follow the order? Would not get scrambled eggs!

Can we express this more generally?

Making Scrambled Eggs for 10

- Ingredients:

  20 eggs, 1 tbsp oil, salt

- Instructions:

  Step 1: Add oil to pan.
  Step 2: Heat pan on stove.
  Step 3: Crack 1st egg into pan.
  Step 4: Crack 2nd egg into pan.
  …
  Step 22: Crack 20th egg into pan.
  Step 23: Add salt.
  Step 24: Mix until light and cooked.

- Output:

  Scrambled eggs for 10!

Another way of making eggs for 10

Repeat 10 times:

- Ingredients:

  2 eggs, 1 tbsp oil, salt

- Instructions:

  Step 1: Add oil to pan.
  Step 2: Heat pan on stove.
  Step 3: Crack 1st egg into pan.
  Step 4: Crack 2nd egg into pan.
  Step 5: Add salt.
  Step 6: Mix until light and cooked.

- Output:

  Scrambled eggs!

Done repeating.

Making Scrambled Eggs for "N" people

- Ingredients:

  2*N eggs, 1 tbsp oil, salt ← Introduce input **variable**, N

- Instructions:

  Step: Add oil to pan.

  Step: Heat pan on stove.

  Step: For i = 1 to 2*N ← Introduce a **loop**, with count **variable**, i

  Step: Crack i^{th} egg into pan. ← Repeat step, until count variable is done

  Terminate loop when counter reaches max

  Step: End loop

  Step: Add salt.

  Step: Mix until light and cooked.

- Output:

  Scrambled eggs for N !

Variables

- Variables are containers (in memory) for information that you want to store.
- The information you put in the variable is called its value.
- Putting a value in the container is called assigning a variable.
- When you write the variable name in a program, the variable name will be replaced by its value, whenever the computer gets to that point in the program. This is called accessing a variable.
- Variables can be reassigned values at different points in the program. (When that happens, the old value is lost.)

Naming variables (and functions)

- What names can we choose for our variables?
- Lots of possibilities! But it depends on the programming language.
- Avoid key words/characters that are used by the programming language to mean something specific.
  - E.g. “for” is used to denote a loop type, so can’t use it to name a variable!
  - Same for mathematical operators and most punctuation signs.
  - Can’t have a space (“ “) as part of the name, because spaces denote a change of word.

  These are just examples - specific rules change from one programming language to another.
- Good names: Bob, C3P0, The_cat_in_the_hat
- Bad names: 1*2, a-b, for, print, if, while

Types of variables

- Boolean: Value can be 0 or 1.

  E.g. Bob = 0

- Integer: Value can be any whole number.

  E.g. Bob = 5

- Float: Value can be any real number (up to some pre-set precision).

  E.g. Bob = 1.3333333333

- Characters: Value can be any ASCII character.

  E.g. Bob = “p”

- Strings: Value is a sequence of characters.

  E.g. Bob = “Happy birthday!”

Worrying about types of variables

- The types listed in the previous slide are available in most programming languages. Other types are possible in some languages.
- Most languages require that the variable type be declared before the variable is assigned or accessed.

  E.g. integer bob

  bob = 5

  bob = bob + 2

- Some languages do not handle variable types (e.g. Scratch).
  - Means that the language assumes all variables are of the same type, and each variable is stored in a memory container of a fixed size.
- Some languages do not require type to be declared (e.g. Matlab).
  - Also means the language will assume something about variable type, and reserve a memory block accordingly.

Arithmetic Operations

- The arithmetic operators (+, -, *, /) are reserved for mathematical operations.

  Let ***x*** be a variable.

  x = 5 <- Initial variable assignment.

  x = x+2 <- Variable is re-assigned.

- What happens if you skip the first step?
  - It depends on the programming language.
  - In some cases, it may depend on what was in the memory block associated with *x* before! Could be bad….
- Most languages also accommodate logical variables and operators.

Comparison Operators

- The comparison operators compare two values (numbers or variables).
- There are several comparison operators:

  < *less than*

  <= *less than or equal to*

  > *greater than*

  >= *greater than or equal to*

  == *equal to*

  != *not equal to*

Loops

- Syntax for telling the computer to repeat the same instruction many times.
- Example: Write a program to sum the numbers from 1 to K.

SumUpTo (K)	Program name "SumUpTo" with input variable "K"
n = 1	Add new variable "n" to count from 1 to K
sum = 0	Add new variable "sum" to store the intermediate sum
While n <= K	Loop command "While", with the termination defined by comparison "n<=K"
sum = sum + n	First instruction in the loop.
n = n+1	Second instruction in the loop.
End loop	Syntax requires that you specify up to which instruction goes inside the loop
PrintText "The sum of numbers 1 to K is"	Print the result.
PrintVariableValue "sum"	

- What if you want to print the intermediate results? Easy to do!

Comparing Loops

- Different types of loops:

```
SumUpTo ( K )
n =  1
sum = 0
While  n <= K
        sum = sum + n
        n = n+1
End loop

PrintText “The sum of numbers 1 to K is”
PrintVariableValue “sum”
```

```
SumUpTo ( K )

sum = 0
For n = 1 to K
        sum = sum + n
End loop

PrintText “The sum of numbers 1 to K is”
PrintVariableValue “sum”
```

- Similar syntax, slightly different functionality. In this case, it does not matter, but in other cases, it might.

Loop that quits when it reaches its goal

- Consider summing integers, until the sum reaches 100.

```
SumUpTo (  )
n =  1
sum = 0
While  sum <= 100
        sum = sum + n
        n = n+1
End loop

PrintText “The sum of numbers  is”
PrintVariableValue “sum”
```

- Can we do the same thing for any target max value? Yes!

```
SumUpTo ( max )
n =  1
sum = 0
While  sum <= max
        sum = sum + n
        n = n+1
End loop

PrintText “The sum of numbers  is”
PrintVariableValue “sum”
```

- This is much easier to do with a While loop than with a For loop.

Functions

```
SumUpTo ( K )
n =  1
sum = 0
While  n <= K
        sum = Add (sum, n)
        n = n+1
End loop
PrintText "The sum of numbers  is"
PrintVariableValue "sum"
```

```
Add ( n1, n2)
 sum = n1 + n2
Return sum
```

- Here both AddToSum(K) and Add(n1, n2) are examples of functions.
- Functions allow you to write re-usable code.
- When a function is called, the computer "jumps" to the body of the function to execute that block of code, then comes to where it left off.
- Functions can call other functions. E.g. **AddToSum()** calls **Add()**

Functions

```
SumUpTo ( K )
n =  1
sum = 0
While  n <= K
        sum = Add (sum, n)
        n = n+1
End loop
PrintText "The sum of numbers  is"
PrintVariableValue "sum"
```

```
Add ( n1, n2)
 sum = n1 + n2
Return sum
```

- Functions can take inputs (variables or constants). E.g. **K**, **n1**, **n2, 100.**
- Functions only know variables which are given as input, or defined inside.

  E.g. **Add()** does not know about variable **n**.
- Functions can return outputs.

Functions that call themselves

- A really compact way of summing up the first K integers:

```
SumUpTo ( K )
If  K > 0
        sum = SumUpTo ( K -1) + K
Else
        PrintText “The sum of
numbers  is”
        PrintVariableValue “sum”
End conditional
```

- New element here: **conditional statement**

Conditional statement

- Execute a block of code only **if** a certain statement is true.

General form:

```
If  some expression is true
        Do this.
Else
        Do that.
```

Generalizing some of these ideas

- You can have conditionals within loops, conditionals within functions, conditionals within conditionals, functions within loops, functions within conditionals, functions within functions, loops within loops, loops within functions, loops within conditionals ...

- Most examples we saw today deal with numbers. Many programs deal with other types of data, e.g. Strings, logical variables, etc.
  - Easy to declare variables of those types.
  - But: Need to be careful about how we use mathematical operators and comparison operators when dealing with these types of variables.

Is your program correct?

- Very important to have the correct program!
- Throughout the software industry, roughly 90% of efforts goes to testing/debugging and only 10% of efforts for programming.

- Can you "check" for correctness of the program?
  - Need to check it works for all possible inputs.
  - E.g. Check that it won't loop forever. Check that it won't set variables to wrong values.

- Substantial work in this area in software engineering.

Take-home message

- Know the difference between a programming language and a program.
- Understand the need to be very precise when writing instructions for a computer.
  - But realize that there are different ways of instructing the computer to do the same thing.
- Understand the basic concepts of programming:
  - Variables, mathematical operators, comparison operators, loops, conditionals, functions.

Final comments

- Some material from these slides was taken from:
  - *http://cim.mcgill.ca/~sveta/COMP102/*