CS187 - Science Gateway Seminar for CS and Math

Fall 2013 – Class 6

Sep. 19, 2013

Programming and Programming Languages – Why, How and What?

- A programming language is a **formal language** (vs. natural language) that communicates a set of instructions to a machine – computer for exmaple.
- Programs tell the computer what to do (even the operating system is a big, complicated program!), express and implement algorithms.
- A set of instructions a machine can understand.

Machine Language vs. Human Language

- The first "programmable machines" precede modern computers (remember last week's talk!).
- The first modern computer languages – early 1950s.
- First programs used machine language – directly understood by the CPU.
- The first compiler – 1952 (Grace M. Hopper).
- Compiler – "translates" programs into machine language (itself a program!)

Assembly Language	Machine Code
add $t1, t2, $t3	04CB: 0000 0100 1100 1011
addi $t2, $t3, 60	16BC: 0001 0110 1011 1100
and $t3, $t1, $t2	0299: 0000 0010 1001 1001
andi $t3, $t1, 5	22C5: 0010 0010 1100 0101
beq $t1, $t2, 4	3444: 0011 0100 0100 0100
bne $t1, $t2, 4	4444: 0100 0100 0100 0100
j 0x50	F032: 1111 0000 0011 0010
lw $t1, 16($s1)	5A50: 0101 1010 0101 0000
nop	0005: 0000 0000 0000 0101
nor $t3, $t1, $t2	029E: 0000 0010 1001 1110
or $t3, $t1, $t2	029A: 0000 0010 1001 1010
ori $t3, $t1, 10	62CA: 0110 0010 1100 1010
ssl $t2, $t1, 2	0455: 0000 0100 0101 0101
srl $t2, $t1, 1	0457: 0000 0100 0101 0111
sw $t1, 16($t0)	7050: 0111 0000 0101 0000
sub $t2, $t1, $t0	0214: 0000 0010 0001 0100

Assembly and Machine language code

Why Not Machine Language, then?

- See previous slide...
- People are not good at machine language
- Modern (high level languages) are the middle ground between human and computer languages
- They are easier to understand, to debug and to move between different computers

```
1
2   public void bubbleSortA(int[] nums, int size)
3
4   {
5
6   for(last = size-1; last > 0; last = last - 1)
7       {
8           for(current = 0; current < last; current = current + 1)
9               {
10                  if (nums[current] < nums[current + 1])
11                  {
12                      temp = nums[current];
13                      nums[current] = nums[current+1];
14                      nums[current+1] = temp;
15                  }
16              }
17      }
18  }
```

Why Not Natural Language, then?

- All natural languages are believed to be Turing complete
- That is – they can express anything a Turing machine can
- Most programming languages are Turing complete too.
- It follows – all human languages are equivalent to one another (in the Computability Theory sense)
- But... human languages have context, expressions and ambiguity:
  - Time flies like an arrow
  - I have been chasing my own tail
  - Give me the key
- Not everyone speaks the same language/dialects
- Machine translation between natural languages can be atrocious...

Different, But Similar

- A computer language is also a language.
- It has semantics and syntax and a vocabulary.
- You already speak at least one language... so you should not be afraid of programming languages.
- It's actually like learning another, much simpler language.
- A program is simply a list of unambiguous, clear instructions given to a machine that cannot understand context.

Chocolate Cake Example

Input

Algorithm

Recipe
CHOCOLATE CAKE

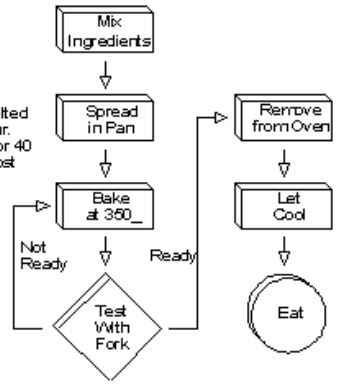
4 oz. chocolate	3 eggs
1 cup butter	1 tsp. vanilla
2 cups sugar	1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

Declare variables:
chocolate, butter, sugar, eggs, vanilla, flour, mix

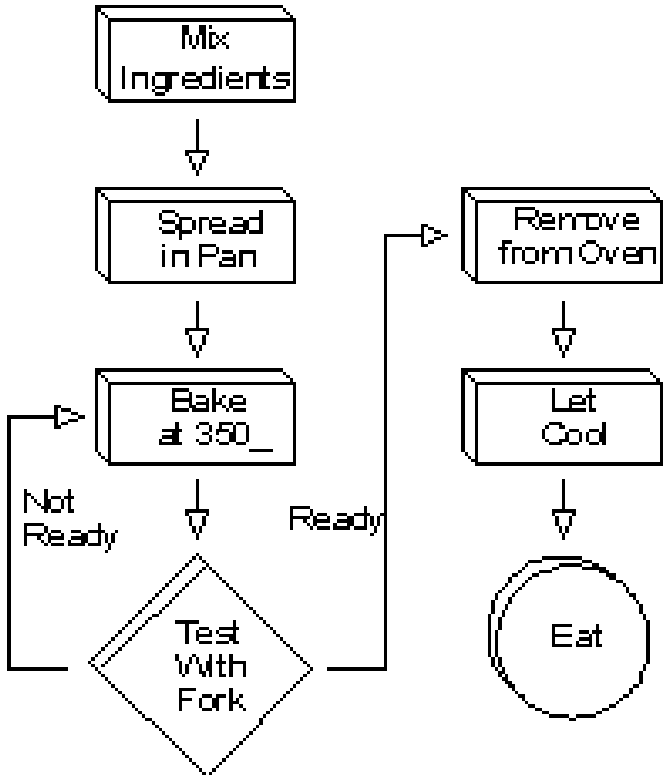
mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
bake (mix, 350)



Output

Chocolate Cake Example

- Must be unambiguous, solve the problem and terminate.
- One starting point and (one or more) end point(s).
- Input and output (both optional).
- Control flow – we follow the instructions (not necessarily in the order of their appearance) and at any stage we're at some "block".
- Conditional branching – if...then... (else...).
- Loops – repeat some actions for a certain number of times or until some condition is filled.



More Relevant Example, 3-Way Maximum

- **Problem:** Given 3 numbers X, Y, Z – find the largest and print on the screen.
- **Input:** 3 numbers
- **Algorithm:** Calculate the maximum of 3 numbers
- **Output:** The largest number

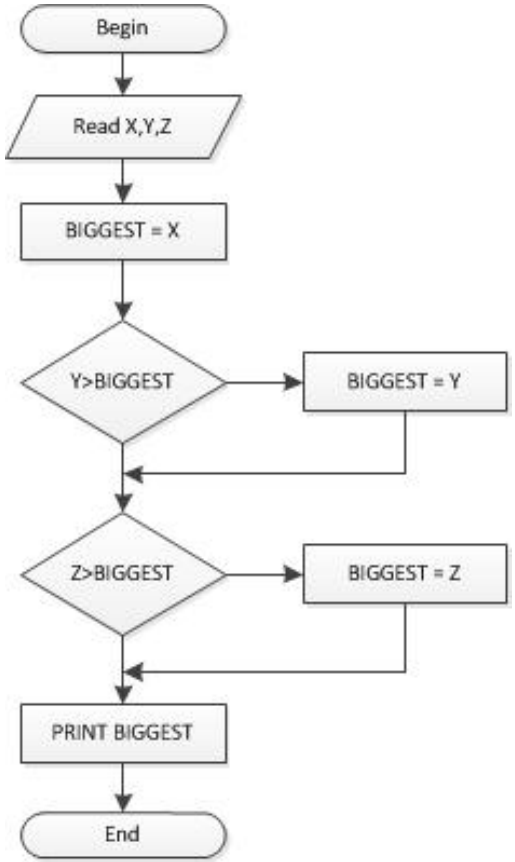
3-Way Maximum in Plain English

Input: X, Y, Z

1. Not knowing better, assume X is the largest and denote it "biggest" (the largest we have so far).
2. Compare X and Y. If Y is larger than X, then now Y is the "biggest".
3. Otherwise, do nothing (leave X as "biggest").
4. Compare Z and "biggest".
5. Return the largest of the two as the maximum.

Try to run it in your head with an actual example (say – 17,8,29) to convince yourselves that it works. Remember that the computer is a machine and that the program should be able to handle any 3 numbers, not just this specific example.

3-Way Maximum as Flowchart



3-Way Maximum in Java

```
void max3(int x,int y, int z)
{
    int biggest;
    if(x > y) biggest=x;
    else biggest=y;
    if (z > biggest) biggest =z;
    System.out.println(biggest);
}
```

"Pseudocode" – Something in Between

Algorithm 1 Sort (list L of N numbers)

1: Repeat steps (2-4) N times:
2: $m = Minimum(L)$
3: $print(m)$
4: Remove m from L

Algorithm 2 Minimum(list L of M numbers)

1: $tmpmin =$ first element in L
2: **for** Each of the remaining elements in L **do**
3: $i =$ Next element in L
4: **if** $i < tmpmin$ **then**
5: $tmpmin = i$
6: **end if**
7: **end for**
8: **return** $tmpmin$