Lecture 6: More on Loops, Stepwise refinement revisited
Loops: for vs. while

• A for loop is generally used when we know how many times we want to execute some statements.
  i.e. read in n integers and add them up (convention: start at 0)
  
  ```c
  for(i = 0; i < n; i++){
    scanf("%i", &x);
    sum += x;
  }
  ```

• A while loop is generally used when we are not sure how many times we want to repeat.
  i.e. read in positive integers until negative integer is reached, and add them up
  
  ```c
  while(x >= 0){
    sum += x;
    scanf("%i", &x);
  }
  ```
A do-loop or a while-loop is often used to make sure the user inputs valid data.

```c
    do {
        printf("Enter a positive number : ");
        scanf("%i", &x);
    } while(x <= 0);
```
A for loop can be thought of as a while loop

• Recall syntax of for loop
  ```
  int c;
  for (c = 0; c < 10; c++)
  { statements; }
  ```

• This could have been written using while
  ```
  int c = 0;
  while(c < 10)
  {
    statements;
    c++;  }
  ```
Nested Iteration

• Can have loop inside a loop

```c
int i, j;

for(i = 0; i < n; i++)
    for(j = 0; j < m; j++)
        printf("%i %i\n", i, j);
```

• What is the value of count after these nested loops?

```c
int i, j, count = 0;
for(i = 0; i < n; i++)
    for(j = 0; j < m; j++)
        count++;
```
Nested Iteration and Selection

• Can nest loop-statement inside if-statement

    if (n > 0)
        while(i < n){
            printf("*" pedido);
            i++;
        }

• Can nest if-statement inside loop-statement

    /* print even numbers between 0 and n */
    for(i = 0; i < n; i++)
        if(i % 2 == 0)
            printf("%d\n", i);
How are the subtasks in an algorithm combined?

- They can be combined as a **sequence**: first execute task1, then execute task2.
- They can be combined using **selection**: check a condition, if the condition is true execute task1, if not execute task 2.
- They can be combined using **repetition** (aka **iteration**): repeat task1 until a condition becomes true.

Structured programming languages (like C) have language constructs for composing tasks using sequence, selection and repetition.

- We call these **control structures** since they control the execution order of the statements.
C has 7 basic control structures

- Sequence structure: statements listed sequentially are executed sequentially
  
  ```
  a = 4;
  b = 5;
  ```

- 3 selection structures:
  - if
  - if/else
  - switch

- 3 repetition (loop) structures:
  - while
  - for
  - do/while
C’s 7 control structures as flowcharts

- **Sequence**
- **Iteration**
- **Selection**
Recall: Stepwise Refinement

- Stepwise refinement is a divide and conquer approach

1. Clearly state the intended task

2. Divide the task to a set of subtasks and re-express the intended task as an equivalent structure of properly connected subtasks, each solving part of the problem

3. Divide each subtasks far enough until the complexity of each subtask is manageable (i.e., you know how to write a program segment for that subtask)
Rule 1: start with the simplest flowchart

- One rectangle
- A good (and widely applicable) example:
  
  \textit{get some data, calculate and show some results}

- Really just a way to start; clarifies the “big picture”
Rule #2: Sequential decomposition

- Replace a rectangle with two rectangles in sequence
- This “stacking rule” can apply repeatedly: one $\rightarrow$ two, two $\rightarrow$ three, … For example:
  1. Get data
  2. Process
  3. Show results
Rule #3: Nesting

- This “nesting rule” also applies repeatedly, as each control structure has rectangles
- e.g., nest a `while` loop in an `if` structure:
  ```c
  if (n > 0)
     while (i < n)
        printf("%d ", i++);
  ```
Rule #4: apply #2 or #3 repeatedly

• Stack, nest, stack, nest, nest, stack, ... gets more and more detailed as one proceeds
  – Think of control structures as building blocks that can be combined in two ways only.
  – Captures the essence of stepwise refinement: keep adding details as they arise
    • Means add control structures as long as they are needed

• Top-down design: start with forest, do trees later