Concepts of Programming

Instructor: Nihshanka Debroy
Polya’s Method -- How to Solve It
1. Understand the problem
2. Devise a plan
3. Carry out the plan
4. Look back

The plan you make to solve the problem is called an **algorithm**

A computer **program** is an expression of an algorithm in a computer language

**Programming** enables us to use the computer as a problem solving tool
Definition: program

- General
  - a series of steps to be carried out or goals to be accomplished
    - for example, a program of study

- Computer science
  - a sequence of instructions a computer can interpret and execute that tells the computer how to perform a specific task or directs its behavior
Pseudocode

- Shorthand for specifying algorithms
- Leaves out implementation details
- Leaves in essence of algorithm

Example:

```plaintext
procedure Greetings

    Count = 3

    while(Count >0) do this:
        {print the message “Hello” and Count=Count-1}

```

- What does this algorithm do?
- How many times does it print “Hello”? 
Sequential search

procedure Search (List, TargetValue)
if (List empty)
  then
    (Declare search a failure)
else
  (Select the first entry in List to be TestEntry;
  while (TargetValue > TestEntry and
         there remain entries to be considered)
    do (Select the next entry in List as TestEntry.);
  if (TargetValue = TestEntry)
    then (Declare search a success.)
  else (Declare search a failure.)
  ) end if
Picking courses

1. Make a list of courses you want to register for, in order of priority
2. Start with empty schedule. Number of courses = 0.
3. Choose highest priority class on list.
4. If the chosen class is not full and its class time does not conflict with classes already scheduled, then register for the class (2 steps):
   1. Add the class to the schedule
   2. Increment the number of classes scheduled

5. Cross that class off of your list.
6. Repeat steps 3 through 5 until the number of classes scheduled is >= 4, or until all classes have been crossed out.
7. Stop.
Flowcharts

Begin

Make list of classes you want to take

Num Classes = 0

Choose highest priority class on list

Is this class full?

yes

Is there a time conflict?

no

Add the class to your schedule. Increment Num Classes.

no

Cross the class off your list.

yes

Num Classes >= 4?

no

More classes on list?

yes

no

End
Describing an Algorithm

Pictures
Plain English
Pseudo-code (almost a programming language)
Programming Language

More easily expressed → More precise
Data organization

- Data organization
  - representation of input, output, intermediate values
    - intermediate values hold information derived from input or other intermediate values that we want to remember for later on
      - assignment of names to values, which may assume different values or remain constant
    - The names we assign to values are called variables
  - A variable type describes the values it can take on
    - such as integer (int) or boolean (boolean)
Assignment statements

- Variables are assigned values using assignment statements.
- Assign the value of an expression to a variable:
  \[ \langle \text{variable} \rangle = \langle \text{expression} \rangle \]

- Variables that appear on the right side of an assignment statement must have previously defined values.
- The value resulting from evaluation of the expression is assigned to the variable on the left side of the equation.
Examples

- The equals sign should be interpreted as "is assigned the value of" or "is replaced by"

\[ \pi = 3.14159; \]
\[ x = 15; \]
\[ y = 30; \]
\[ z = x + y; \]
\[ a = 80; \]
\[ b = 90; \]
\[ \text{average} = (a + b) / 2; \]

- Variables with previously assigned values can appear on both sides of the assignment statement

\[ \text{sum} = 0; \]
\[ \text{sum} = \text{sum} + 1; \]
Control structures

- Determine flow of execution of a program's instructions
  - Sequential execution
    - instructions follow one another in a logical progression
  - Selective execution
    - provides a choice depending upon whether a logical expression is true or false
  - Repetitive execution
    - the same sequence of instructions is to be repeated a number of times
- We can construct any algorithm using combinations of control structures
Sequential execution

statement 1

statement 2

\vdots

statement n
Selective execution

- Allows program to take alternate logical paths
- Decisions are based on the value of a logical expression
  - logical expressions evaluate to true or false
- Relational operators are used to make comparisons in a logical expression
  
  \[
  \text{==}, \text{!=}, \text{<}, \text{<=}, \text{>}, \text{>=}
  \]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>5==5</td>
<td>true</td>
</tr>
<tr>
<td>5!=5</td>
<td>false</td>
</tr>
<tr>
<td>3&lt;8</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2&gt;=8</td>
<td>false</td>
</tr>
<tr>
<td>7!=5</td>
<td>true</td>
</tr>
<tr>
<td>9==1</td>
<td>false</td>
</tr>
</tbody>
</table>
if-else selection

- `expression` followed by:
  - `false` leading to `statements2`
  - `true` leading to `statements1`
Selective execution: if else-if else

- if else-if else
  - distinguish between three or more cases
  - example: convert numerical grade to A-F
- If a logical expression is true, the remainder of the statements are bypassed
  - good design - check likeliest cases first

```plaintext
if(average ≥ 90) 
  grade = A;
else if(average ≥ 80) 
  grade = B;
else if(average ≥ 70) 
  grade = C;
else if(average ≥ 60) 
  grade = D;
else
  grade = F;
```
Repeating code

- Repeating code is bad
- Writing repetitive code is tedious
- Debugging repetitive code is hard
- Avoid repeating code through:
  - Subroutines/methods
  - Loops
Repetitive execution: for-loop

- Repetition controlled by a **counter**
- Statements executed once for each value of a variable in a specified range
  - start and end values are **known**
- Initial statement: assign start value of counter
- Test: logical expression comparing counter to end value
- Update statement: assign new value to counter

```c
for( initial-statement ; test ; update-statement )
{
    ... statements ...
}
```
Example: for-loop

```c
for (k=a; k<b; k++)
{
    x=x+k;
}
```

- `k` is the counter variable
- `a`, `b`, `x` must have assigned values
- `k++` increments `k` by one

- If `a=3`, `b=7`, and `x=10` prior to loop execution, what is the value of `x` when the loop terminates?
Minimum of two integers

- Analyze the problem
  - Inputs
    - $x$ first integer
    - $y$ second integer
  - Output
    - $\text{min}$ minimum of $x$ and $y$
  - How do we find the minimum??
    - if the first number is smaller than the second number, then the first number is the minimum
    - else, the second number is the minimum
Minimum of two integers

- Design an algorithm to solve the problem

1. Get input values for \( x \) and \( y \)
2. Compute minimum value

   ```
   if(x < y)
   min = x;
   else
   min = y;
   ```

3. Return output value \( \text{min} \)
Sum of positive integers

- Analyze the problem
  - Input
    - n a positive integer
  - Output
    - sum sum of all positive integers \(\leq n\)
  - How to find the sum??
    - \(\text{sum} = 1+2+3+ \ldots +n\)
    - initialize \(\text{sum}=0\)
    - let \(k\) loop over the values \([1, n]\)
    - compute \(\text{sum} = \text{sum} + k\) at each iteration of loop
Sum of positive integers

- Design an algorithm to solve the problem

1. Get input value for $n$
2. Compute sum of integers 1 through $n$

```java
sum=0;
for(k=1; k<=n; k++)
{
    sum=sum+k;
}
```

3. Return output value sum
1. Design an algorithm to compute the inclusive sum between two integers
   - example: for the input values 2 and 6, your algorithm should output 20 (because \(2+3+4+5+6 = 20\))

2. Design an algorithm that computes \(x^n\) for an integer \(x\) and a non-negative integer \(n\)
   - \(x^n\) is defined as follows:
     \[
x^0 = 1 \quad \text{if} \ n=0, \ \text{otherwise}
     
x^n = x \times x \times x \times x \times x \times \cdots \times x
\]
   - \(n\) times
3. Design an algorithm to compute the maximum of two integers

4. Design an algorithm to compute the inclusive product between two integers
   - example: for the input values 3 and 6, your algorithm should output 360 (because $3 \times 4 \times 5 \times 6 = 360$)
What is a Programming Language?

- A vocabulary
- A set of syntactical (grammatical) rules
- Instructs the computer to perform specific tasks
- Can do almost anything in a programming language
- Particular languages encourage you to do things a certain way
- Examples: C, C++, Java, Perl