Algorithm Design

**Review**

- 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: algorithm**

- Well-defined computational procedure that takes some value or set of values as **input** and produces some value or set of values as **output**
  - that is, a sequence of computational steps that transform the input into the output
- Can also view an algorithm as a tool for solving a well-specified **computational problem**
  - the problem statement specifies in general terms the desired input/output relationship
  - the algorithm describes a specific computational procedure for achieving that relationship

CLRS, Introduction to Algorithms

---

**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

---

**Stages of program development**

1. Problem analysis and specification
2. Data organization and algorithm design
3. Program coding
4. Execution and testing
5. Program maintenance

---

**Problem specification and analysis**

- **Specification**
  - description of the problem’s input
    - what information is given and which items are important in solving the problem
  - description of the problem’s output
    - what information must be produced to solve the problem
- **Analysis**
  - generalize specification to solve given problem and related problems of same kind
  - divide complex problems into subproblems
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:
  $$<\text{variable}> = <\text{expression}>$$
- 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 assignment statement

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;
average=(a+b)/2;
```

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

```
sum=0;
sum=sum+1;
```

Algorithm design and refinement

- Basic description of an algorithm
  - get input values
  - compute output values for the given input
  - return output values
- Algorithm refinement
  - adding problem specific details
  - computation of output values
- Check correctness of algorithm after steps are designed
  - sample data
  - mathematical analysis

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
...
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
  \[ \text{==, !=, <, <=, >, >=} \]

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

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

Selective execution: if

- Executes statements when the logical expression is true
- Multiple statements are enclosed in curly braces { } otherwise only the first statement following the if is executed
- If logical expression is false statements are not executed – computer proceeds to next statement in the program

```
if( logical-expression )
{
  ... statements ...
}
```

if selection

```
expression
  true → statements

false
```

Selective execution: if-else

- Executes one set of statements when logical expression is true and different set of statements when expression is false
- Used to select between two cases
- Multiple statements are enclosed in curly braces

```
if( logical-expression )
{
  ... statements1 ...
}
else
{
  ... statements2 ...
}
```

if-else selection

```
expression
  true → statements1

false
```

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

```
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;
```
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

\[
\text{for( initial-statement; test; update-statement )}
\{ 
  \ldots \text{ statements } \ldots 
\}
\]

Example: for-loop

\[
\text{for(k=a; k<b; k++)}
\{ 
  x=x+k; 
\}
\]

• If \(a=3\), \(b=7\), and \(x=10\) prior to loop execution, what is the value of \(x\) when the loop terminates?

for-loop repetition

Repetitive execution: while-loops

• Repetition controlled by a logical expression
  – statements executed while the logical expression is \text{true}, loop exits when logical expression is \text{false}
  – some variable in the logical expression must change with each repetition
    • otherwise, we loop forever!

\[
\text{while( logical-expression )}
\{ 
  \ldots \text{ statements } \ldots 
\}
\]

Example: while-loops

\[
k=a; 
\text{while}(k<b)
\{ 
  x=x+k; 
  k=k+1; 
\}
\]

• If \(a=3\), \(b=7\), and \(x=10\) prior to loop execution, what is the value of \(x\) when the loop terminates?
**For-loop or while-loop?**

- When to use a for-loop:
  - always for counting!
  - you know how many times to execute the loop
- When to use a while-loop:
  - number of repetitions needed is unknown

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

```c
k=a;
while(k<b)
{
  x=x+k;
  k=k+1;
}
```

A for loop can always be written as a while loop.

**Shampoo algorithm**

- Q: Why did the computer scientist die in the shower?
- A: He followed the instructions on the shampoo bottle.

- Problem with shampoo algorithm:
  - no terminating condition!
- Shampoo algorithm has what is called an infinite loop
- How to fix?

**Detecting infinite loops**

- Problem: compute sum of positive integers ≤ n
- Assume n is an input value. Are the following while-loops correct or incorrect? Why?

```c
sum=0;
while(n>0)
{
  sum=sum+n;
  n=n-1;
}
```

```c
sum=0;
while(n>0)
{
  sum=sum+n;
  n=n+1;
}
```

```c
k=1;
while(sum<n)
{
  sum=sum+k;
  k=k+1;
}
```