Introduction to High-Level Language Programming
Chapter 7

Pseudo-code vs. Assembly

Disadvantages of Assembly

- The programmer must manage movement of data items between memory locations and the ALU.
- Programmer must take a "microscopic" view of a task, breaking it down to manipulate individual memory locations.
- Assembly language is machine-specific.
- Statements are not English-like (Pseudo-code)
### Pseudo-code vs. High-level Programs

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>High-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set <code>sum</code> to 0</td>
<td><code>void main()</code></td>
</tr>
<tr>
<td>Set <code>i</code> to 1</td>
<td><code>{</code></td>
</tr>
<tr>
<td>While <code>i ≤ 5</code> do</td>
<td><code>int i, sum, N;</code></td>
</tr>
<tr>
<td>Get value for <code>N</code></td>
<td><code>sum = 0;</code></td>
</tr>
<tr>
<td>Add <code>N</code> to <code>sum</code></td>
<td><code>i = 1;</code></td>
</tr>
<tr>
<td>Increase value of <code>i</code> by 1</td>
<td><code>while (i &lt;= 5) {</code></td>
</tr>
<tr>
<td>End loop</td>
<td><code>cin &gt;&gt; N;</code></td>
</tr>
<tr>
<td>Print the value of <code>sum</code></td>
<td><code>sum = sum + N;</code></td>
</tr>
<tr>
<td></td>
<td><code>i = i + 1;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
<tr>
<td></td>
<td><code>cout &lt;&lt; sum;</code></td>
</tr>
</tbody>
</table>

### High-level Programming Languages

- The programmer need not manage the details of the movement of data items between memory and ALU.
  - Doesn’t even have know there is a register in the ALU for performing arithmetic.
- The programmer has more macroscopic view of a task, using less primitive building blocks
  - E.g. doesn’t work with individual memory locations anymore.
- High-level languages are portable.
  - Same program can run on different architectures.
- More English (pseudo-code) like!

### Program Translation

```
A=B+C
LOAD
STORE
Compiler
Assembler
Linker
Loader
0101
0101
0101
0110
0101
1001
1001
0110
```

---

**Chapter 7: High-level Language Programming**
The C++ Programming Language

• We will use (a subset of) the C++ programming language to introduce you to programming in a high-level language.
• Although the syntax differ from one programming language to the next, the basic concepts apply to all (most) high-level languages.
• C++ is an object-oriented language
  – although we will not learn about that in this course
  – but you can learn all about it in CMPUT114!

Example C++ Program

// Program Numerology. This program gets the user's favorite number and prints a greeting.
#include <iostream.h>
void main()
{
  int your_number;
  cout << "Please enter your favorite number:";
  cin >> your_number;
  cout << endl;
  cout << "Your favorite number is " << your_number << "." << endl;
  cout << "That is a nice number." << endl;
}

General C++ Program Structure

Prologue comment [optional]
Include directives [optional]
Functions [optional]
Main function
{
  Declarations [optional]
  Body
}
Virtual Data Storage (Data items)

- One improvement of a high-level language is to make data manipulation easier.
  - J: DATA -1 -- tedious in an assembly!
  - LOAD J and STORE J
- Instead of working with individual memory locations (as in assembly), we work with more abstraction in form of data items.
- In the program we give English like names to data items to identify them.

Identifiers

- Names in programs are called identifiers.
- An identifier can consist of any combination of letters, digits, and _, except:
  - cannot start with a digit
  - cannot be same name as a C++ keyword.
- Should try to use descriptive names
- Identifier are case-sensitive, for example
  - a and A do refer to different data items
Example of identifiers

- Legitimate names:
  - cmput101, My1stCar
  - A, b
  - Your_Guess, Number_Of_Homeruns
  - A_speling_mistake

- Not legitimate names (Why?):
  - 1stCar
  - int
  - Iwin!
  - arrgg@#!#t

Data items

- Store data used in program:
  - read in from user (Get / In ...)
  - constants used in program (N: .DATA 5)

- A data item can be declared either as a constant or a variable.
  - Constants are initialized with a value, but their value cannot be changed after that.
  - The value of a variable can be changed as needed.

- The keyword const in the declaration indicates that the data item is a constant.

Declaration of data items.

- We need to declare data items in our program prior to using them.
- The declaration tells:
  - whether the data item is a constant or a variable.
  - the identifier that will be used in the program to name the data item.
  - the data type for the data item.
Standard Data Types in C++

- Following are examples of predefined data types used in C++:
  - There are more basic data types.
  - Programmers can create their own types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>an integer number (e.g. 10, -5)</td>
</tr>
<tr>
<td>double</td>
<td>a real number (e.g. 3.1415, 2.1)</td>
</tr>
<tr>
<td>char</td>
<td>a character (e.g. 'a', 'C')</td>
</tr>
</tbody>
</table>

Example

```cpp
void main()
{
    // Declaring a constant.
    const double PI = 3.1416;

    // Single variable declared at a time.
    int my_number;
    double GPA;
    char initial_letter;

    // Can declare many data-items of the same type together.
    int height, base;
}
```

Example

```cpp
void main()
{
    // Declaring constants
    const int MIN_VALUE = 0;
    const int MAX_VALUE; // Error

    MIN_VALUE = 45; // Error

    cout << "MIN_VALUE is now " << MIN_VALUE;
}
```
Statement Types

- Three different kinds of statements:
  - Input/Output (I/O) Statements
  - Assignment Statements
  - Control Statements
- Notes:
  - An executable statement ends with a ; (semi-colon).
  - Can split one statement between lines!
  - Comments: // Indicates that the rest of the line is a comment.

Input/Output Statements

- In algorithms:
  - Get value of A
  - Print value of A
- In assembly:
  - IN A
  - OUT A
- In C++:
  - cin >> A;
  - cout << A;

Input Statement

- The input statement reads a value from the input stream (keyboard) into a variable

  \[
  \text{cin} \gg \text{your\_number};
  \]

- Upon entering the input statement the program stops and waits for the user to enter a value, e.g.

  \[
  24 <\text{enter}>
  \]

  The variable \text{your\_number} now contains the value 24
Output Statement

- The output statement writes a value of a variable(s) to the output stream (screen)
  
  ```cpp
  cout << your_number;
  ```

- We can write more than one value at a time:

  ```cpp
  cout << "Your number is " << your_number << endl;
  ```

Special considerations

- We need to include the compiler directive
  
  ```cpp
  #include <iostream.h>
  ```
  to tell in which library the `cin` and `cout` commands are.

- When printing text we enclose it within " ", e.g.
  - `cout << "My lucky number is: " << endl;`
  - `endl` forces a line-break on the screen

Example program using I/O

- Let us look at our example program again

  ```cpp
  // Example Program Using I/O.
  #include <iostream.h>
  void main()
  {
      int your_number;
      cout << "Please enter your favorite number:"
      cin >> your_number;
      cout << endl;
      cout << "Your favorite number is " << your_number << endl;
      cout << "That is a nice number." << endl;
  }
  ```
Output when we run the program

Please enter your favorite number:
2 4 <enter>
Your favorite number is 24
That is a nice number.

The Assignment Statement

• The assignment statement assigns a value to a program variable.
• General format in C++:

\[
<\text{variable}> = <\text{expression}>;
\]

The expression to the right gets evaluated, and the result is written into the memory location referenced to by the variable.

Examples of assignments

```c
void main()
{
    int A, B, C;
    int my_number, your_number, our_number;
    A = 0;
    B = -2;
    C = (A-B) / B + (2*B) ;
    ...
    my_number = 5;
    your_number = 3;
    our_number = my_number + your_number;
    ...
}
```
### Arithmetic Operations in Expressions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbol</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>( C = A + B; )</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>( C = A - B; )</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>( C = A \times B; )</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>( C = A / B; )</td>
</tr>
</tbody>
</table>

### A Practice Problem

Write a program that calculates the area of a triangle, given its height and base.

\[
A = \frac{\text{height} \times \text{base}}{2}
\]

#### Write the algorithm in pseudo-code:

1. Get values for `height` and `base`
2. Set value of area to \((\text{height} \times \text{base})/2\)
3. Print value of area

#### C++ code

```cpp
// This program calculates the area of a triangle, given its height and base.
#include <iostream.h>
void main()
{
    double area, height, base;
    cout << "Enter the height of the triangle:";
    cin >> height;
    cout << "Enter the base of the triangle:";
    cin >> base;
    area = (height * base) / 2;  // Note parentheses!
    cout << " The area of the triangle is " << area << endl;
}
```
Running Our Program

Enter the height of the triangle: 2
Enter the base of the triangle: 4
The area of the triangle is 4

Enter the height of the triangle: 10
Enter the base of the triangle: 5
The area of the triangle is 25

Control Flow Statements

• We have three types of a control flow in a program:
  – Sequential
  – Conditional
  – Looping

Sequential Flow of Control

• The default case.
• No special commands needed.
Example: Sequential Flow in C++

```cpp
// An example of sequential flow.
#include <iostream.h>
void main()
{
    int your_number;
    cout << "Please enter a number:";
    cin >> your_number;
    cout << "Your number is " << your_number << "." << endl;
}
```

Example Program Output

Please enter a number: 2
Your number is 2.

Please enter a number: 5
Your number is 5.

Conditional Flow of Control

- Begins with evaluating a Boolean condition.
- If condition is true, then execute statement S1.
- Otherwise, if condition is false, execute statement S2.
- In both cases, statement S3 is executed next.
If-else Statement in C++

```cpp
if ( <boolean expression> )
    <statement-1>;
else
    <statement-2>;
```

Example: Conditional Flow in C++

```cpp
// An example of conditional flow.
#include <iostream.h>
void main()
{
    const int lucky_number = 8;
    int your_number;
    cout << "Please guess my lucky number:";
    cin >> your_number;
    cout << "Your number is " << your_number << ".
    if ( your_number == lucky_number ) // boolean expression
        cout << "You win!";
    else
        cout << "You lose!";
}
```

Example Program Output

Please, guess my lucky number: 2
Your number is 2.
You lose!

Please, guess my lucky number: 8
Your number is 8.
You win!
Multi-way If-else Statement

```cpp
if ( <condition> )
...
else if ( <condition> )
...
else if ( <condition> )
else
...
```

Boolean Conditions (Expressions)

- Expression can be either true or false.

<table>
<thead>
<tr>
<th>Expression</th>
<th>A=0; B=1;</th>
<th>A=1; B=2;</th>
</tr>
</thead>
<tbody>
<tr>
<td>A == 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A != B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(A+1) &lt; B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C++ Comparison Operators

<table>
<thead>
<tr>
<th>The same value as</th>
<th>== 2 == 5</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than</td>
<td>&lt; 2 &lt; 5</td>
<td>true</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;= 5 &lt;= 5</td>
<td>true</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt; 2 &gt; 5</td>
<td>false</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;= 2 &gt;= 5</td>
<td>false</td>
</tr>
<tr>
<td>Not the same value as</td>
<td>!= 2 != 5</td>
<td>true</td>
</tr>
</tbody>
</table>
Examples: Comparison Operators

```c++
if ( your_number == 8 )
    cout << "You win!";
else
    cout << "You lose!";

if ( your_weight_lbs > your_ideal_weight_lbs )
    cout << "You need to diet!";
else
    cout << "More ice-cream?";
```

C++ Boolean Operators

- Boolean operators can be used to make more complex Boolean expressions.

<table>
<thead>
<tr>
<th></th>
<th>&amp;&amp;</th>
<th>(2&lt;5) &amp; (2&gt;7)</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>!</td>
<td>!(2==5)</td>
<td>true</td>
</tr>
<tr>
<td>NOT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples: Boolean Operators

```c++
if ( (your_number > 5) && (your_number<8) )
    cout << "You win!";
else
    cout << "You lose!";

if ( (your_weight < your_lower_limit_weight) ||
    (your_weight > your_upper_limit_weight) )
    cout << "See your doctor about your weight.";
else
    cout << "You are in a good shape.";
```
Compound Statements

- What if we want to execute more than one statement within a if-statement?
  - We can group arbitrary many statements together by enclosing them within {

{
    <statement-1>
    <statement-2>
}

Example: Compound statements

```cpp
if ( (your_number > 5) && (your_number<8) )
{
    cout << "You win!";
    cout << "Guess you got lucky!";
}
else
{
    cout << "You lose!";
    cout << "You'll never guess the right number!";
}
```

// Example program using a compound if-else statement.
#include <iostream.h>
void main()
{
    int your_number, my_number;
    cout << "Please enter a positive number:";
    cin >> your_number;
    if ( your_number >= 0 )   // need to use a compound form
    {
        my_number = 2 * your_number;
        cout << "My number is " << my_number;
    }
    else // not necessary to use a compound form.
    {
        cout << "Sorry, your number is negative!" << endl;
    }
}
Looping Flow of Control (while)

- Begins with evaluating a Boolean condition.
- While condition is true execute statement S1 and then re-evaluate Boolean condition. Repeat until ...
- ... condition is false, then go to statement S2.

While-loop Statement in C++

```
while ( <Boolean expression> )

   <statement-1>;

while ( <Boolean expression> )

   {<statement-1>;
    :
    ;
    <statement-n>;}
```

Examples of while loops

```
while ( M >= 1 )
    M = M - 2;

while ( i <= 5 )
    { cout << "Enter a grade: ";
      cin >> grade;
      total = total + grade;
      i = i + 1;
    }`
What does this program print out?

```cpp
#include <iostream.h>
void main()
{
    int number;
    number = 1;
    while ( number > 0 )
    {
        cout << number << endl;
        number = number - 1;
    }
    cout << number;
}
```

What is the output?

```cpp
#include <iostream.h>
void main()
{
    int number;
    number = 1;
    while ( number > 0 )
    {
        cout << number << endl;
        number = number - 1;
    }
    cout << number;
}
```

Example #1: Program Output

Please, guess my lucky number: 2
Sorry, enter another number: 6
Sorry, enter another number: 9
Sorry, enter another number: 8
You guessed 8, my lucky number!

// Example #1: Use of the while statement.
// The user guesses the program's lucky number!
#include <iostream.h>
void main()
{
    const int lucky_number = 8;
    int your_number;
    cout << "Please, guess my lucky number: ";
    cin >> your_number;
    while ( your_number != lucky_number )
    {
        cout << "Sorry, enter another number: ";
        cin >> your_number;
    }
    cout << "You guessed " << lucky_number << ", my lucky number!";
}
// Example #2: Use of the while-statement.
// The user enters a number, and the program divides
// the number in half while it is greater or equal to one,
// printing out all the intermediate results.
#include <iostream.h>

void main()
{
    int number;
    cout << "Enter a number: ";
    cin >> number;
    while ( number >= 1 )
    {
        cout << number << endl;
        number = number / 2;
    }

Example #2: Program Output

Enter a number: 40
40
20
10
5
2
1

// Example #3: Use of the while statement.
// The program determines if a given number is odd or even.
#include <iostream.h>
void main()
{
    int number;
    cout << "Enter a positive number: ";
    cin >> number;
    while ( number >= 1 )
    {
        number = number - 2;
    }
    if ( number == 0 )
        cout << "The number is even.");
    else
        cout << "The number is odd.");
Example #3: Program Output

Enter a positive number: 4
The number is even.

Enter a positive number: 7
The number is odd.

Practice Problem 1

• Write a program that calculates the user's GPA. Before entering the grades the user first enters how many grades there are.

Get a value for \( N \), the number of courses
Set the value of total to 0
Set the value of \( i \) to 1
While \( i \leq N \) do
Get a value for grade
Set total to \( ( \text{total} + \text{grade} ) \)
Increase the value of \( i \) by 1
End loop
Set the value of GPA to \( \frac{\text{total}}{N} \)
Print the value of GPA

// Example #4: This program calculates GPA.
#include <iostream.h>
void main()
{   
    int i, N;
    double grade, GPA, total;
    total = 0.0;
    cout << "Enter the number of courses taken: ";
    cin >> N;
    i = 1;
    while ( i <= N ) {  
        cout << "Enter a grade: ";
        cin >> grade;
        total = total + grade;
        i = i + 1;
    }
    GPA = total / N;
    cout << "The GPA is " << GPA << endl;
}
Practice Problem 1: Program Output

Enter the number of courses taken: 5
Enter a grade: 5
Enter a grade: 7
Enter a grade: 8
Enter a grade: 5
Enter a grade: 8
The GPA is 6.6

Practice Problem 2 (take one)

• Write a program that reads in 5 integers and prints out the numbers that are larger than the last number entered (i.e. the fifth number).

Get values for N_1, N_2, ..., N_5
Set i to 1
While i < 5 do
  If N_i > N_5 then
    Print N_i
  Increase i by 1
End loop

• How do we write this algorithm in C++?

The Array Data Type

• An array groups together a collection of data items of the same type, e.g.

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>
```

• In a C++ program we:
  – Specify the size of the array when we declare it.
  – Use an index in the range 0, ..., size-1 to refer to individual elements in the array.
Arrays in C++

```cpp
#include <iostream.h>
void main()
{
    int grade[10]; // Declaring an array of 10 integers.
    int i;
    grade[0] = 9;
    grade[1] = 6;
    ...
    grade[9] = 8;
    i = 0; // Note: indexing range is from 0 ... 9
    while ( i < 10 ) {
        cout << grade[ i ] << endl;
        i = i + 1;
    }
}
```

Arrays in C++

```cpp
// This program reads in 5 integers and stores them in an array.
#include <iostream.h>
void main()
{
    const int MAX = 5; // The number of integers to read in.
    int a[MAX];
    int n, i;
    i = 0;
    while ( i < MAX ) {
        cout << "Enter a number: ";
        cin >> n; // Note: Why not cin >> a[ i ]? Limitation in lab-software!
        a[ i ] = n;
        i = i + 1;
    }
}
```

Practice Problem 2 (take two)

- Write a program that reads in 5 integers and prints out the numbers that are larger than the last number entered (i.e. the fifth number).

Get values for $N_1$, $N_2$, ..., $N_5$

Set $i$ to 1

While $i < 5$ do

If $N_i > N_5$ then

Print $N_i$

Increase $i$ by 1

End loop
```cpp
#include <iostream.h>

void main()
{
  const int MAX = 5;  // Number of values to read in.
  int i, n, N[MAX];
  // Read in the integers; use a loop!
  cout << "Enter the numbers: " << endl;
  i = 0;
  while (i < MAX) {
    cin >> n;
    N[i] = n;
    ++i;  // Note: same as i = i + 1;
  }
  // Print out the numbers that are larger than the last (fifth) number.
  cout << "Larger than last:"
  i = 0;   // Remember to reset i
  while (i < MAX-1) { // Note: MAX-1 !
    if (N[i] > N[MAX-1])
      cout << " " << N[i];
    i = i + 1;
  }
}
```

**Practice Problem 2: Program Output**

Enter the numbers:
2
5
1
8
4

Larger than last: 5 8

**Repeat Loops**

- What if we use a Repeat loop in the algorithm?

  Get values for N₁, N₂, ..., N₄
  Set i to 1
  Repeat until i ≥ 5 do
    If Nᵢ > N₄ then
      Print Nᵢ
    Increase i by 1
  End loop

- How do we code Repeat loops in C++?
Repeat vs. While

- We can always rewrite a Repeat as a While loop
  - C++ has a loop similar to Repeat, but we will not look at that in this course.

<table>
<thead>
<tr>
<th>Repeat</th>
<th>While</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get values for ( N_1, N_2, \ldots, N_5 )</td>
<td>Get values for ( N_1, N_2, \ldots, N_5 )</td>
</tr>
<tr>
<td>Set ( i ) to 1</td>
<td>Set ( i ) to 1</td>
</tr>
<tr>
<td>Repeat until ( i \geq 5 ) do</td>
<td>While ( i &lt; 5 ) do</td>
</tr>
<tr>
<td>( \text{If } N_i &gt; N_5 \text{ then} )</td>
<td>( \text{If } N_i &gt; N_5 \text{ then} )</td>
</tr>
<tr>
<td>( \text{Print } N_i )</td>
<td>( \text{Print } N_i )</td>
</tr>
<tr>
<td>Increase ( i ) by 1</td>
<td>Increase ( i ) by 1</td>
</tr>
<tr>
<td>End loop</td>
<td>End loop</td>
</tr>
</tbody>
</table>

Elements Correctness and Style

- Important to make our programs correct:
  - Logically correct (do what supposed to do)
  - Syntactically correct (so can compile)
- Also, important to make them readable (why?):
  - No more than one statement in each line.
  - Proper indentation.
  - Descriptive identifier names.
  - Documentation (comments).

What does this program do?

```cpp
#include <iostream.h>

void main() {
    int x; int q10; x=1; cout << "Enter a number: "; cin >> q10;
    while ( q10 > 1 ) {
        x = x * q10;
        q10 = q10 - 1;
    }
    cout << x; }
```
```cpp
#include <iostream.h>
void main()
{
    int x;
    int q10;
    x = 1;
    cout << "Enter a number: ";
    cin >> q10;
    while ( q10 > 1 )
    {
        x = x * q10;
        q10 = q10 - 1;
    }
    cout << x;
}
```

### Proper Indentation
```cpp
#include <iostream.h>
void main()
{
    int x;
    int q10;
    x = 1;
    cout << "Enter a number: ";
    cin >> q10;
    while ( q10 > 1 )
    {
        x = x * q10;
        q10 = q10 - 1;
    }
    cout << x;
}
```

### Descriptive Identifier Names
```cpp
#include <iostream.h>
void main()
{
    int factorial;
    int n;
    factorial = 1;
    cout << "Enter a number: ";
    cin >> n;
    while ( n > 1 )
    {
        factorial = factorial * n;
        n = n - 1;
    }
    cout << factorial;
}
```
// Given a number n the program outputs n factorial, e.g.
// 6! = 6 * 5 * 4 * (n-1) * (n-2) * ... * 2 * 1
#include <iostream.h>
void main()
{
    int factorial;
    int n;
    factorial = 1; // 0! = 1
    cout << "Enter a number: ";
    cin >> n;
    while ( n > 1 )
    {
        factorial = factorial * n;
        n = n - 1;
    }
    cout << factorial;
}