Introduction to High-Level Language Programming

Chapter 7

Pseudo-code vs. Assembly

Set *sum* to 0
Set *i* to 1
While $i \leq 5$ do
  Get value for *N*
  Add *N* to *sum*
  Increase value of *i* by 1
End loop
Print the value of *sum*

```
.BEGIN  -- Sum 5 numbers
Loop:   LOAD      Five
        COMPARE   i
        JUMPGT    Done
        IN        N
        LOAD      sum
        ADD       N
        STORE     sum
        INCREMENT i
        JUMP      Loop
Done:   OUT       sum
        HALT
Five:   .DATA     5
i:      .DATA     1
sum:    .DATA     0
N:      .DATA     0
.END
```


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

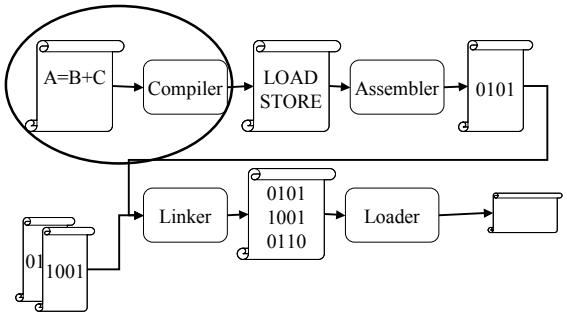
Set *sum* to 0
Set *i* to 1
While $i \leq 5$ do
  Get value for *N*
  Add *N* to *sum*
  Increase value of *i* by 1
End loop
Print the value of *sum*

```
void main()
{
  int i, sum, N;
  sum = 0;
  i = 1;
  while ( i <= 5 ) {
      cin >> N;
      sum = sum + N;
      i = i + 1;
  }
  cout << sum;
}
```


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



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 !

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 7

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;
}
```

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 8

General C++ Program Structure

```
Prologue comment          [optional]
Include directives        [optional]
Functions                 [optional]
Main function
{
   Declarations           [optional]
   Body
}
```

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 9

Structure of Example Program

```
Prologue comment
Include directives
Functions
Main function
{
  Declarations
  Body
}
```

```
// 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 " << …
 cout << "That is a nice number." << endl;
}
```

CMPUT101 Introduction to Computing

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.

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 11

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

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 12

Example of identifiers

- Legitimate names:
  - cmput101, My1stCar
  - A, b
  - Your_Guess, Number_Of_Homeruns
  - A_speling_mistake
- Not legitimate names (Why?)
  - 1stCar
  - int
  - Iwin!
  - arrrgg@#!#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.

int	an integer number (e.g. 10, -5).
double	a real number (e.g. 3.1415, 2.1).
char	a character (e.g. ‘a’, ‘C’).

Example

```
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

```
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 kind 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.

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 19

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;

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 20

Input Statement

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

```
cin >> your_number;
```

- Upon entering the input statement the program stops and waits for the user to enter a value, e.g. 24 <enter>

  The variable *your_number* now contains the value 24

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 21

Output Statement

- The output statement writes a value of a variable(s) to the output stream (screen)

```
cout << your_number;
```

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

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

Your number is 24

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 22

Special considerations

- We need to include the compiler directive

```
#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

CMPUT101 Introduction to Computing (c) Yngvi Bjornsson 23

Example program using I/O

- Let us look at our example program again

```
// 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++:

  <variable> = <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

```
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;
  ...
}
```

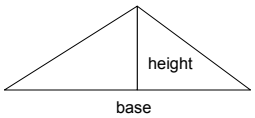
Arithmetic Operations in Expressions

Addition	+	C = A + B;
Subtraction	-	C = A – B;
Multiplication	*	C = A * B;
Division	/	C = A / B;

A Practice Problem

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

A = (height x base) / 2



Write the algorithm in pseudo-code:

Get values for *height* and *base*
Set value of area to (*height***base*)/2
Print value of *area*

C++ code

```
// 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

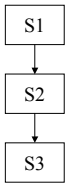
Enter the height of the triangle: 2
Enter the base of the triangle: 4
The area of the triangle is 4

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++

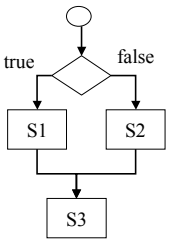
```
// An example of sequential flow.

#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++

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

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


Example: Conditional Flow in C++

```
// 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 << "." << endl;
  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

```
if ( <condition> )
        …
else if ( <condition> )
        …
else if ( <condition> )
        …
else
        …
```


Boolean Conditions (Expressions)

- Expression can be either *true* or *false*.

Expression	A=0; B=1;	A=1; B=2;
A == 0		
A != B		
(A+1) < B		

C++ Comparison Operators

The same value as	==	2 == 5	false
Less than	<	2 < 5	true
Less than or equal to	<=	5 <= 5	true
Greater than	>	2 > 5	false
Greater than or equal to	>=	2 >= 5	false
Not the same value as	!=	2 != 5	true

Examples: Comparison Operators

```
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.

AND	&&	(2<5) && (2>7)	false
OR	||	(2<5) || (2>7)	true
NOT	!	!(2==5)	true

Examples: Boolean Operators

```
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>;
}
```

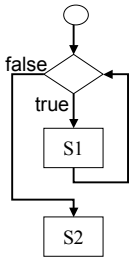

Example: Compound statements

```
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?

```
// What is the output?
#include <iostream.h>
void main()
{
  int number;
  number = 1;
  while ( number > 0 )
    cout << number << endl;
    number = number - 1;
  cout << number;
}
```

```
// What is the output?
#include <iostream.h>
void main()
{
  int number;
  number = 1;
  while ( number > 0 )
  {
     cout << number << endl;
     number = number - 1;
  }
  cout << 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 #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 #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 ≤ N do
   Get a value for grade
   Set total to ( total + grade )
   Increase the value of i by 1
End loop
Set the value of  GPA  to ( 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 N1, N2, ..., N5
Set i to 1
While i < 5 do
  If Ni > N5 then
        Print Ni
  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.

0	1	2	3	4	5	6	7	8	9
5	6	1	9	4	5	3	8	10	5

- 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++

```
#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++

```
// 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 N1, N2, ..., N5
Set i to 1
While i < 5 do
  If Ni > N5 then
        Print Ni
  Increase i by 1
End loop
```

```
#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, ..., N5
Set i to 1
Repeat until i ≥ 5 do
  If Ni > N5 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.

```
Get values for N1, N2, ..., N5
Set i to 1
Repeat until i ≥ 5 do
  If Ni > N5 then
      Print Ni
  Increase i by 1
End loop
```

```
Get values for N1, N2, ..., N5
Set i to 1
While i < 5 do
  If Ni > N5 then
      Print Ni
  Increase i by 1
End loop
```


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?

```
#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; }
```

Continued ... One statement each line

```
#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;
}
```

Continued ... Proper indentation

```
#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;
}
```

Continued ... Descriptive identifier names

```
#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;
}
```

Continued ... Documentation added.

```
// Given a number n the program outputs n factorial, e.g.
//  n! = n * (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;
}
```