

COMSC-110 Chapter 3

Values, Variables, and Calculations

Valerie Colber, MBA, PMP, SCPM

You should know

- How to edit and save C++ files.
- Compile and run your programs.
- You may change your choice of compilers or systems during this course.
- C++ programmers ought to be able to work independently of any specific system, editor, or compiler.
- For now, find something that works and stick with it.

Algorithm

- An effective method for solving a problem using a finite sequence of instructions.
- Programmers must first develop the algorithm they will use BEFORE they can write the program!
- Think of algorithm instructions like a recipe.
- The recipe contains:
 1. Description of the result or objective
 2. What INPUT, PROCESSING, OUTPUT, and DATA are needed.
 3. The step by step instructions for achieving that objective.
 4. What to test to make sure the algorithm works under all circumstances.

Algorithm parts

1. **Objective** of what is to be done
2. **Requirements/specifications** (what you need, like formulas, units of measure, format of numbers, dates, currency, etc., for INPUT, PROCESSING, OUTPUT, DATA)
3. **Algorithm** with unique identifiers for each instruction
4. **Test cases**: You need to have more than one test case. Make one test case a "benchmark" test case where you know what the result ought to be. Then make the next test case a comparison case with different input values to make sure it works. Other test cases might be for unexpected circumstances during execution.

Algorithm

- Is programming language independent.
- Tells another programmer what the program will do during execution.
- Does NOT tell another programmer HOW to write the program.
- Another programmer ought to be able to write the same program you did just from the algorithm alone!

What are the 3 kinds of errors?

- Syntax – violating the rules of a programming language
- Logic – telling the user of an algorithm or the computer how to do the objective incorrectly.
- Run time/Execution – when the computer tries to execute an instruction that is something the computer cannot do, such as divide by 0.

What every programmer needs to do

- As a programmer, you need to create or choose the algorithm to achieve the result or objective of the program in order to develop your program.
- You need to Play Computer to debug your program.

How to Play Computer

- Keep track of what variables and their data values the program uses in RAM.
- Keep track of program output.
- As you be the computer and execute each instruction in the program.

Code blocks

- Usually it takes more than a single statement to do one thing in a program.
- Similarly in written language it can take several sentences to complete a thought.
- In written English we use paragraphs to group related sentences together to form a thought.
- In programming we use "code blocks".

Code blocks

- Code blocks are written in programs purely for organizational purposes.
- Their use, correct or otherwise, has no impact on a program's operation.
- Code blocks usually contain a comment line that explains the purpose of the code block, so that a reader of the program does not have to read all the statements in the code block in order to figure out what it does.
- Short code blocks do not always need a comment line, if they are easy to read and understand.
- To make code blocks easier to spot in a long program, they are usually separated by skipped lines.

How programs store and use data during execution

- Applications share and use RAM (random access memory) to store and use data during execution.
- Programmers use “variables” to allocate space in RAM during the execution of the program to store the data used by the program during execution.
- The visuals in the lecture notes and in your book show you how a programmer visualizes RAM because hardware is different, so the amount of RAM available on any computer can differ, and a programmer has to know how much RAM their application has available to use.
- So while the computer maps RAM by memory addresses, your program can use your variable’s identifier (name) to map to that memory address(es) in RAM to locate that data.

Values, Variable, Calculations

- **Values** are data values.
- **Variables** are for storing data values during the execution of a program.
- **Calculations** process data.
- This covers the three parts of simple programs:
 1. **establishing input values**
 2. **performing calculations**
 3. **displaying the results**

Values

- A **value** is a piece of information that can be used in a computer program.
- Values can be combined in such simple operations as addition, or processed in more complicated operations like spell-checking.
- Values can also be displayed to a user (output), entered by a user (input), or read from a disk file (input).
- There are two distinct types of values that are used in computer programs: *numbers* and *text* (or words).

Values

- The ways in which computers remember, process, and display data values depends on whether the values are numeric or text.
- **Numbers** can be either **whole numbers**, as might be used in counting (like 1, 2, 3, etc.), or they can allow **fraction or decimal parts**, as might be used to record grade point average (for example, 3.24).
- **Text** can be either single **character** (i.e.: “a” or “A”), or they can be multiple characters called a **string** of characters (i.e.: “321 Golf Club Rd”).

Values by another name

- Data values
- Literal values
- Hard coded values
- Programmer's input
- Assigned values
- Fixed values

Categories of data

- Numbers
 - Text
 - Graphics
 - Audio
 - Video
- Data categories are in order of **how much storage they require**
 - There are **certain efficiencies in the ways that computers deal with the simpler forms of numbers and text.**

Variables

- A **variable** is something in a computer program that programmers use to store a value (in random access memory (RAM) during program execution).
- Data values can be specified in a program's code.
- Data values can be input from a keyboard or text file.
- Data values can be the result of a calculation.
- **Variables are the things that store data values in a computer program.**
- **Variables are reusable.**
- **The number of variables that a programmer can use in a program is basically limited by the amount of memory in the computer, so for today's desktop computers, it is virtually unlimited.**

Identifiers

- Variables have names, or "identifiers".
- By referring to the name of a variable in a computer program, its associated data can be stored or retrieved.
- Variable names are chosen by the programmer.
- Variable names must consist of one or more lowercase letters, uppercase letters, digits, and/or underscore symbols -- no spaces are allowed!
- Variable names may not begin with a digit, and it cannot be the same as one of the hundred or so words used in the programming language (called "keywords"), such as "if" and "while".

Identifiers

- Besides the rules, there are some *conventions* which most programmers follow: variables usually begin with a lowercase letter.
- Identifiers consisting of more than one word, since they cannot be separated by spaces, are run together with the first character in each word after the first being uppercase.

Declaration Statements

- For each item of data to be stored, there needs to be a variable.
- To create a variable in a C++ program, you have to specify two things:
 1. a unique identifier for the variable
 2. *the type of data to store* in the variable.
- This specification must be made as a *statement*, and it is called a *declaration statement*.
- Statements usually end in a semicolon.
- To reserve memory to store a numeric value in a variable named "age": `int age;`

Data Types

- Integer
 - Double
 - Float
 - Character
 - String
- `int` is for storing whole numbers, like 12 or 0 or -99
 - `double` is for storing floating point numbers, like 1.5 or 100.0
 - `char` is for storing 1-character text, such as 'T' or 'F' (true or false), or 'A', 'B',... (grades), or digits '0', '1',..., or punctuation symbols '.', ' ', '!', '...',...
 - `string` is for storing multiple character text, such as 'T' or 'F' (true or false), or 'A', 'B',... (grades), or digits '0', '1',..., or punctuation symbols '.', ' ', '!', '...',...
 - String requires `#include <string>` and `using namespace std;` at the top of the program

Examples of C++ data types

Some Data Types In C++

int	for storing whole numbers, like 12 or 0 or -99
double	for storing floating point numbers, like 1.5 or 100.0
char	for storing 1-character text, such as 'T' or 'F' (true or false), or 'A', 'B',... (grades), or digits '0', '1',..., or punctuation symbols ';', '.',...
string	for storing text of any length, such as "I am learning how to program!" (requires <code>#include <string></code> and <code>using namespace std;</code> at the top of the code)

- In C++ a number is more defined as to whether it is an integer (whole numbers) or a floating point number with decimals.
- Text is a string data type in C++ and a single text character is the char data type in C++.

Declaration Statements

- Data types are *keywords* in the C++ computer language.
- Note that most C++ textbooks use the keyword *float* for single-precision floating point numbers, instead of *double*.
- The reason for using "double" instead of "float" is that it is more directly compatible with the *math "library"*.

Where variable declarations go

```
int main()
{
    // data
    variable declarations go here

    //output introduction
    ....

} //main
```


Example declarations

```
#include <string>
using namespace std;
```

```
int main()
{
    //data
    int age;
    double birthWeight;
    string firstName;
    char gender;
}
```

- Note the indenting of the lines containing the declaration statements.
- Indenting is not important to the compiler -- it is only for human-readability and organization of code.
- Note two new lines were added, beginning with "#include" and "using".
- They are required when using the "string" data type in C++.
- The data type and the identifier are separated by a single space, but it can be more than one if desired for alignment of the code.
- Each declaration statement ends with a semicolon.

Visualizing memory (RAM)

age	21
birthWeight	7.7
firstName	Joe
gender	M

5 basic ways to store a value in a variable

1. by specifying a value upon declaration
2. by assigning a value anytime after declaration
3. by performing a calculation and storing the result
4. by transferring a value from the keyboard
5. by transferring a value from a text file.

Assignment upon declaration

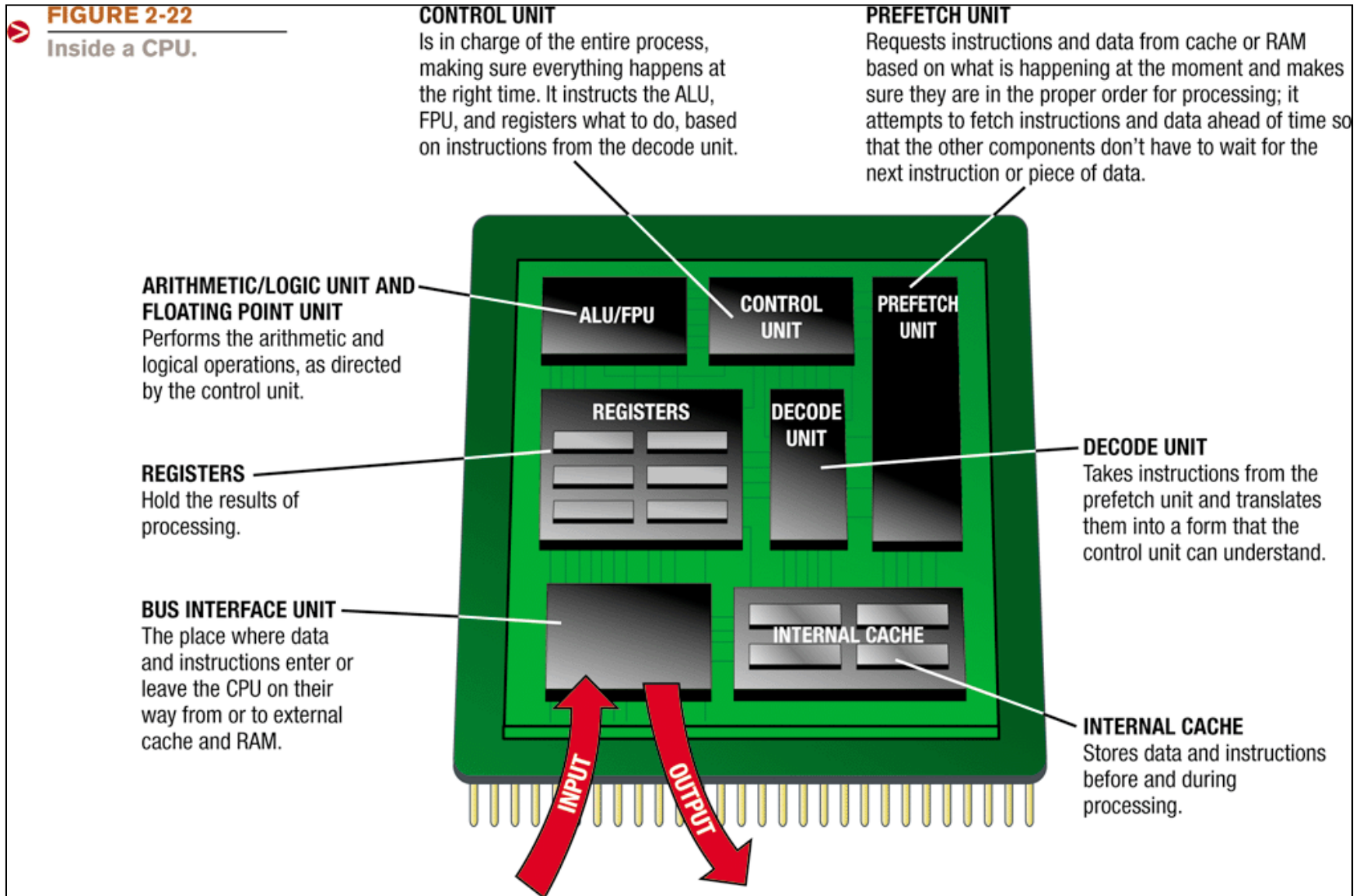
```
int age = 21;
```

- To specify a value to a variable *after* a variable's declaration, use a statement like **age = 21;**, leaving off the data type specification.
- This is known as an *assignment statement*, and it can only be used *after* "age" is declared.
- Per the first control structure, *sequential processing*, it is okay to reassign values to variables as often as you like, whether or not they were initialized upon declaration.
- But you can **only declare a variable once**, because of the uniqueness requirement for identifiers (variable names).

Using variables

- In C++, *you can* use a variable in a calculation or in output before a value is assigned to it, but its uninitialized value is unpredictable and the results will be in error!
- Actually, some compilers (such as Visual C++) produce a warning error if you use uninitialized variables.

Typical CPU components



Arithmetic Operators and the ALU

Order of Precedence

- Parentheses: $()$
- Exponents: $^$
- Multiplication: $*$
- Division: $/$
- Addition: $+$
- Subtraction: $-$
- Modulus: $\%$

which results in the *remainder* of the integer division problem "b goes into a... with the remainder of <result>".

Calculations

- Once there are variables with valid values, they can be considered as inputs to calculations.
- Calculations take place in *expressions*, which usually involve variables, numbers, and an *operation symbol*, like a plus sign.
- In an expression like **a + b**, the **a** and **b** are called *operands*. An operand in a calculation expression can be a variable name, as shown in the examples so far, or they can be values.

Using calculation expressions in statements

- Note that the equal sign in the assignment statement is *not* the same as the equal sign in mathematics!
- In math, it is used to express equivalence between the two sides of the equal sign.
- In C++, it means to evaluate the expression on the *right side* of the equal sign, and store the result in the variable on the *left side* of the equal sign.
- It does *not* establish a equivalence for the computer to remember and apply where appropriate -- remember, sequential processing!
- You cannot put anything except for a single variable on the left side.
- In assignment statements, resolve what is on the *right-hand side of the equal sign* first, then store the resulting value in the variable that appears on the *left-hand side of the equal sign*.
- Never put anything other than a single variable's name to the left of the equal sign

Statements vs Expressions

- **Statement** is a complete programming language instruction
- **Expression** is part of a statement that can use arithmetic and logic operators or parameters in a statement.

expressions vs statements

- Expressions only contain **identifiers**, **literals**, and **operators**, where operators include arithmetic and boolean operators, and can be reduced to some kind of **value**.
- Statements are everything that can make up an **instruction** in a programming language.
- Expressions are parts of a statement.

Using assignment upon declaration

```
int a = 2;  
int b = 2;  
int c = a + b;
```

Using separate declarations and assignment statements

```
int a;  
int b;  
int c;  
a = 2;  
b = 2;  
c = a + b;
```

An application counting

int sum = 0; *stores 0 in variable "sum"*

sum = sum + 1; *replaces 0 with 0+1, or 1*

sum = sum + 1; *replaces 1 with 1+1, or 2*

Visualizing memory and playing computer variable Sum

0

1

2

Complex expressions

- Math operations in C++ involve 2 values.
- But we often have to solve problems with more than 2 values, such as the average of 3 numbers.
- Multiple-step calculations can be done by breaking the steps involving only two values.
- So we can write this in a series of statements, or we can use complex expressions that let us combine a series of 2-value expressions into a single statement.

Using simple expressions

//variables

int age1 = 19; //first age

int age2 = 21; //second age

int age3 = 30; //third age

int averageAge; //average age

averageAge = age1 + age2;

averageAge = averageAge + age3;

averageAge = averageAge / 3;

Using complex expressions

```
int age1 = 19;
```

```
int age2 = 21;
```

```
int age3 = 30;
```

```
int averageAge = ((age1 + age2) + age3) / 3;
```

Visualizing memory and evaluating a complex expression

<u>age1</u>	<u>age2</u>	<u>age3</u>	<u>averageAge</u>
19	21	30	$((19 + 21) + 30) / 3$
			$(40 + 30) / 3$
			$70 / 3$
			23

Manipulating Text

- any-length text can be manipulated in several ways using C++.
- **Text** stored in two separate variables can be combined, one after the other -- this is called **concatenation**.
- Think of the “+” as NOT addition with text, but rather “and”.
- Text operations can be done on a any-length text variable with another text variable (long or single) or a text value.

Manipulating text example algorithm

10 Output “Congratulations, George
Washington! You have been selected to
receive one of 5 valuable prizes!”

20 Stop

Manipulating text example

Concatenation

```
#include <string>
using namespace std;

...
int main()
{
    //data
    string name = "George Washington";
    string s = "Congratulations, ";

    //concatenation
    s = s + name;
    s = s + "! You have been selected to";
    s = s + " receive one of 5 valuable prizes!";
} //main
```

How to translate these statements into English

- Statement:

`s=s+name;`

- Translation into English:

Set string datatype(text) variable s equal to the current value of variable s AND the value of string datatype (text) variable name.

Visualizing memory and playing computer

<u>Name</u>	<u>S</u>
"George Washington"	"Congratulations, "
	"Congratulations, George Washington"
	"Congratulations, George Washington! You have been selected to"
	"Congratulations, George Washington! You have been selected to receive one of 5 valuable prizes!"

Using uninitialized variables

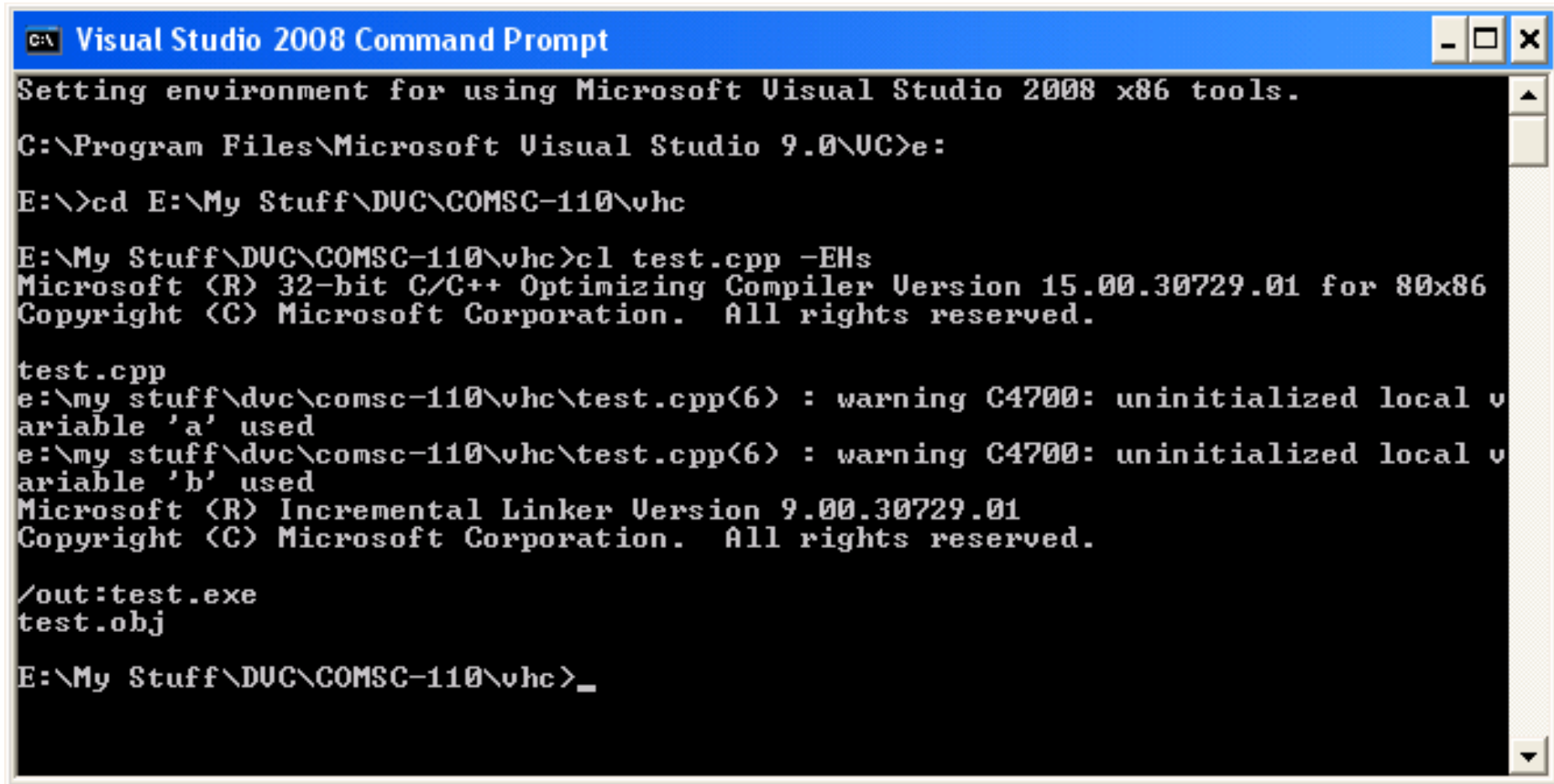
- Values must be stored in numeric variables before they appear in an expression, or else the results will be unpredictable, and usually results in a compiler warning.
- Warnings do not prevent compilation.
- Errors prevent compilation.
- But if you receive a warning, it means that something is wrong and not to expect predictable results from your program.
- So treat warnings as if they were errors, and make corrections as necessary.

Example of using uninitialized variables

```
int main()
{
    int a;
    int b;
    int c;
    c = a + b;

}
```


Example of using uninitialized variables



```
C:\ Visual Studio 2008 Command Prompt
Setting environment for using Microsoft Visual Studio 2008 x86 tools.
C:\Program Files\Microsoft Visual Studio 9.0\VC>e:
E:\>cd E:\My Stuff\DVC\COMSC-110\vhc
E:\My Stuff\DVC\COMSC-110\vhc>cl test.cpp -EHs
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 15.00.30729.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

test.cpp
e:\my stuff\dvc\comsc-110\vhc\test.cpp(6) : warning C4700: uninitialized local v
variable 'a' used
e:\my stuff\dvc\comsc-110\vhc\test.cpp(6) : warning C4700: uninitialized local v
variable 'b' used
Microsoft (R) Incremental Linker Version 9.00.30729.01
Copyright (C) Microsoft Corporation. All rights reserved.

/out:test.exe
test.obj
E:\My Stuff\DVC\COMSC-110\vhc>_
```

Output

- Without output statements, the program keeps all of this information to itself!
- When we refer to "printing" of output, we mean **output** to the console screen, and not to a printer.
- The C++ output statement is **cout << ...;**, where any expression, variable, or number can be substituted for the
- To skip to the next line, use the statement **cout << endl;** -- that's "end-el", not "end-one".
- Output requires the include statement **#include <iostream>**.

Program organization

- Programmer remarks including program objective, programmer, compiler
- Libraries
- Special compiler dependent definitions
- Programmer defined functions
- Main program
- Variable declaration block with programmer remarks for each variable purpose and valid values at the beginning of every function

Example simplistic partial average age algorithm instructions

- 10 First age is 19.
- 20 Second age is 21.
- 30 Third age is 30.
- 40 Set average age to be first age added to second age.
- 50 Add third age to average age.
- 60 Divide average age by 3.
- 70 Output text “the average age is ” and average age.
- 80 Stop

Example: average age (integer)

```
#include <iostream>
using namespace std;

.....
int main()
{
    //data
    int age1 = 19;
    int age2 = 21;
    int age3 = 30;
    int averageAge;

    //processing
    averageAge = age1 + age2;
    averageAge = averageAge + age3;
    averageAge = averageAge / 3;

    //output
    cout << "The average age is ";
    cout << averageAge << endl;
} //main
```

Visualizing memory

<u>age1</u>	<u>age2</u>	<u>age3</u>	<u>averageAge</u>
19	21	30	
			40
			70
			23

Output

The average age is 23

Why is averageAge not 23.33333333?

Example: average age (double)

```
#include <iostream>
using namespace std;

.....
int main()
{
    //data
    int age1 = 19;
    int age2 = 21;
    int age3 = 30;
    double averageAge;

    //processing
    averageAge = age1 + age2;
    averageAge = averageAge + age3;
    averageAge = averageAge / 3.0;

    //output
    cout << "The average age is ";
    cout << averageAge << endl;
} //main
```

Visualizing memory

<u>age1</u>	<u>age2</u>	<u>age3</u>	<u>averageAge</u>
19	21	30	
			40.0
			70.0
			23.333333

Output

The average age is 23.3333333

Why is averageAge 23.33333333?

Example: average age (double)

```
#include <iostream>
using namespace std;
....
int main()
{
    //data
    int age1 = 19;
    int age2 = 21;
    int age3 = 30;
    double averageAge;

    //processing
    averageAge = age1 + age2;
    averageAge = averageAge + age3;
    averageAge = averageAge / 3;

    //output
    cout << "The average age is ";
    cout << averageAge << endl;
} //main
```

Visualizing memory

<u>age1</u>	<u>age2</u>	<u>age3</u>	<u>averageAge</u>
19	21	30	
			40.0
			70.0
			23.0

Output

The average age is 23.0

Why is averageAge 23.0?

Output variable values with labels

- Programmers usually do not print just a variable value all by itself, like

`cout << age << endl;`

because it is difficult to read and understand such output.

- In this case, the output could be something like "21" -- how does the user of your program know what "21" means, especially if it is accompanied by the values of other variables?
- It is better to label your output, so that it looks something like this: "my age is 21" instead of just "21".
- To include a label in an output statement with one variable, use a statement like

`cout << "my age is " << age << endl;`

Correct output of variable's value

```
int age = 21;
```

```
cout << "my age is " << age << endl;
```

- Note the space at the end of the text in quotes -- it separates the label from the variable's value in the printed output.
- When printing a variable's value, be sure to put the *variable's name* in the output statement, and not its *value*

Incorrect output of variable's value

```
int age = 21;
```

```
cout << "my age is 21" << endl;
```

Determining String Length

```
String s;
```

```
s = "Hello";
```

```
s = s + "World";
```

```
int n;
```

```
n = s.length();
```

s is now "HelloWorld"

and n is 10

References

- INTRODUCTION TO PROGRAMMING USING C++ by Robert Burns
- Understanding Computers Today and Tomorrow by Deborah Morley and Charles Parker