

Topic: Introduction to C++

- Higher-level languages
- Programming environments
- A minimal C/C++ program
- Variables, assignments, and initializations
- Input/output streams
- Named constants
- Comments
- The grammar of C++

David Keil 9/03 1

Higher-level languages

- Support control structures and modular decomposition
- Shield programmer from hardware and operating-system details
- Are portable
- Are translated to machine language by compilers or interpreters
- Examples: COBOL, Fortran, Pascal, C, C++, Java

David Keil 9/03 2

Skeleton of a C++ program

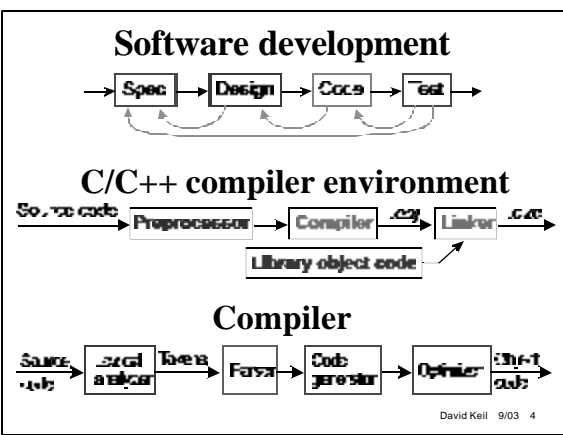
```

// null.cpp:
// Does nothing
void main()
{
}
    
```

// null.cpp: } comments
 // Does nothing }
 void main() } function header
 { } function body (compound statement)

- Every C or C++ program defines a function named *main*

David Keil 9/03 3



An integrated development environment (IDE)

- Editor
- Error diagnostics
- Warnings
- Compiler
- Linker
- Debugger
- Loader
- Help system

David Keil 9/03 5

Using an IDE

- Steps: Code or edit; Build; Execute
- MSVC generates a “console application”
- To correct a syntax error: double-click on error message
- Tip: resist temptation to follow advice of compiler
- For help: press F1 on error message, or highlight a word and press F1
- Remember, one error can trigger many error messages
- Compile on network, not on floppy disk

David Keil 9/03 6

“Work spaces” in MSVC

- The IDE creates a work area for files that are part of a “project” (application)
- Intermediate files and .exe files generated by the compiler and linker go into a subdirectory, *Debug*
- To test a new program, close the work space of the old one
- Later delete any files except .cpp

David Keil 9/03 7

A program with output

```

// hello.cpp:
// Says hello
#include <iostream.h>

void main()
{
    cout << "Hello";
}
    
```

Annotations:

- preprocessor directive: #include <iostream.h>
- output statement: cout << "Hello";
- semicolon terminates simple statement: ;
- standard output stream object: cout
- insertion operator: <<
- string literal: "Hello"

David Keil 9/03 8

A program to add numbers

1. Prompt for integers *input1*, *input2*
2. $sum \leftarrow input1 + input2$
3. Display *sum*

```

// add.cpp
// Prints sum of 2 input integers.
#include <iostream.h>

void main()
{
    cout << "Enter 2 integers to add: ";
    int input1, input2, sum;
    cin >> input1 >> input2;
    sum = input1 + input2;
    cout << input1 << " + " << input2
        << " = " << sum << endl;
}
    
```

Annotations:

- preprocessor directive: #include <iostream.h>
- variable declarations: int input1, input2, sum;
- input statement: cin >> input1 >> input2;
- assignment statement: sum = input1 + input2;
- output statement: cout << ...
- extractor operator: <<

David Keil 9/03 9

Variables and assignments

- A variable is a named data address that occupies space and may be given a value
- Programmer must declare a variable before using it
- A variable has a data type
- A variable gets its value by initialization, assignment, or input
- The assignment operator has a variable to its left and an expression to its right

David Keil 9/03 10

C/C++ statements

- Kinds:
 - variable declarations
 - assignment
 - input/output
 - compound
- Simple statements end with a semicolon (;)
- Compound statements are enclosed by braces

David Keil 9/03 11

Elements of the C and C++ languages

C++ (mid-80s) is an *extension* of C (1971).

David Keil 9/03 12

Assignments and initializations

- An assignment expression has a value:


```
int a,b;      Expression b=2
a = b = 2;    has the value 2
              Assignment operator
```
- An initialization statement is a declaration, and the initialization operator (=) produces no expression:


```
int n = 4;
              Initialization operator
```

David Keil 9/03 13

Input/output

- Input/output (I/O) uses *streams* in C++
- Stream: a sequence of characters going to or from a device
- cout* is the standard output stream object
- cin* is the standard input stream object
- The inserter operator << inserts data into an output stream
- The extractor operator >> extracts data from an input stream and assigns it to one or more variables

David Keil 9/03 14

Standard C and C++ libraries

- Many basic tools, such as for input/output, are in standard *libraries*, not in the language itself
- Standard C++ stream library: *iostream.h*, which declares stream objects like *cin*, *cout*
- To use a standard library, write *#include* and library file name in angle brackets
- Named file is treated as if it were part of your program source file

David Keil 9/03 15

A binary operator yields a value

Operator(s)	Associativity	Example	Value	Side effect
+, -	left	3 - 1 + 4	6	none
=	right	a = b = 1	1	assign
<<, >>	left	cout << a << b cout		I/O

- A binary expression may have both a value and a side effect
- The value may be used in chaining
- Associativity affects result value in chaining

David Keil 9/03 16

Named constants

```
// yearly.cpp
// Tells yearly cost of cable service.
// Uses named constant to store
// the number of months in a year.
#include <iostream.h>
const int MONTHS_IN_YR = 12;

void main()
{
    int per_month = 26; // monthly cost
    cout << "Yearly rate at $" << per_month
        << " per month is $"
        << MONTHS_IN_YR * per_month << endl;
}
```

- Use the *const* keyword when appropriate
- Named constants are reusable, updatable

David Keil 9/03 17

Ways to give a value to a variable

Name	Example	Who chooses
Initialization	int n = 3;	programmer
Assignment	n = 3;	programmer
Input	cin >> n;	user

Giving a value to a constant

- Named constants: one way
Example: const float price = 3.95;
- Literals cannot take new value
Invalid: 3.95 = price;

David Keil 9/03 18

Comments

- `//` starts a comment that lasts to the end of the source line
- `/*` and `*/` delimit a comment
- Comments help make a program readable and understandable
- To debug or maintain a program, one must understand it
- Guideline: write a comment wherever necessary to make programmer's intention clear

David Keil 9/03 19

Writing clear comments

- Not clear:
`// Computes result from input.`
- Not clear:
`/* Displays the absolute value of the difference. */`
- Clear:
`/* This program prompts for 2 integers and displays the absolute value of the difference between them. */`
- A comment is usually a narrative that tells the story of what a program does, or is an assertion about values present as the program runs.

David Keil 9/03 20

Guidelines for formatting source code

- Example:

```
#include <iostream.h>
void main() !
{
    cout << "Hello";
}
```
- Leave an empty line before a function definition such as *main*
- Align pairs of braces vertically
- Indent statements 2-3 spaces

David Keil 9/03 21

Syntax and semantics

- *Syntax* is the set of grammar rules that define a language formally
- *Semantics* is the set of meanings of each of the syntax elements
- The compiler handles a syntax error by halting and displaying a message (usually misleading)
- The compiler follows semantics by generating the appropriate machine code for statements

David Keil 9/03 22

Kinds of tokens (lexical elements)

- keyword (*void*, *main*, *int*, ...)
- identifier (letter or `'_'` followed by a series of letters, digits, `'_'`'s)
- constant literal (numeral, double-quoted string, single quoted character)
- operator (`=`, `+`, `*`, `-`)
- punctuator (semicolon, comma, paren, brace)

Not tokens:

- The compiler ignores white space (space characters, tabs, newlines)
- Compiler ignores comments (`//...`, `/*...*/`)
- Comments and readability are a major factor in effective programming

David Keil 9/03 23

The compiler is case sensitive

- **Total** is a different identifier from **total**
- **Int** is not a keyword

Identifiers and operators

- May not contain spaces,
- Are different from literals:
 “+” is not same as +
“input” is not same as ID **input**
“2” is not same as numeral **2**

David Keil 9/03 24

Specifying grammar rules

- A language is a set of strings, e.g., the set of all possible C++ programs
- A grammar is a set of rules for what is permitted in a language
- C/C++ *tokens* are formed by simple rules; e.g., an integer literal is a series of digits
- Higher-level (*nonterminal*) components (*program*, *statement*, *expression*, etc.) are built from tokens or other nonterminals

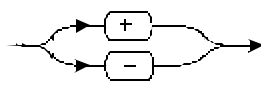
David Keil 9/03 25

Ways to specify syntax

- Plain English (e.g., “A compound statement is a series of statements, in braces”)
- List of alternatives; e.g.:

$$\text{statement-list:}$$

$$\text{statement}$$

$$\text{statement statement-list}$$
- Diagram;
 e.g., *sign*: 

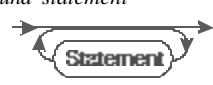
David Keil 9/03 26

Syntax rules and diagrams

compound-statement:
 $\{ \text{statement-list} \}$

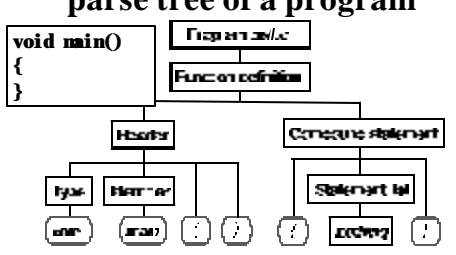
statement-list:
 statement
 $\text{statement statement-list}$

statement:
 nothing
 declaration
 assignment
 IO-statement
 $\text{compound-statement}$

Diagram for *statement-list* 

David Keil 9/03 27

The parser generates a parse tree of a program

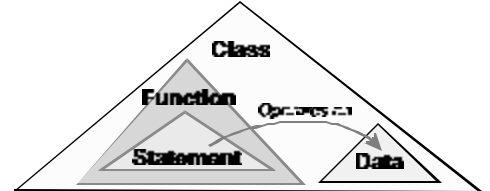


- Each syntax rule is applied by putting a defined element's components under the name of the element

David Keil 9/03 28

Statements, data, functions, objects

How the concepts fit together



David Keil 9/03 29

Virtual machines and applets

- Java compilers may generate *applets* (web applications) that consist of “byte code” rather than machine code
- The “byte code” runs on any browser that supports Java
- The browser contains a *virtual machine* that *interprets* the byte code
- Virtual machine enables distribution of *processor independent* compiled code on the Web

David Keil 9/03 30

Discussion problems

Write a program, with documentation to:

1. Display the words "This is a program" on four lines
2. Initialize and display four integer variables, with values 1, 2, 3, 4
3. As #2, but label output with variable names
4. Input and display four integers
5. Correct this code:

```
int id_num;
cin >> "Enter your ID#: " >>
id_num;
```

David Keil 9/03 31

More discussion problems

1. Display
x
xx
xxx
in *one* statement.
2. Declare named constants for tax as a percentage (5) and discount (33). Display amount due on 4 inputs, consisting of a price and quantity for each of two items. Use no fractions, use integer division if necessary.

David Keil 9/03 32

1. Find *syntax* errors;
2. Find *other* errors

```
#include <iostream.h>
int m;
void()
}
cout >> "Enter 2 #s: ";
cin >> m, n;
int m,n,s,_t1,4f,g.0,n,v+2;
s == m + n, m;
cout << "sum=", s;
{
```

David Keil 9/03 33