



C++ Basics

Brian Malloy, PhD
Department of Computer Science
Clemson University
Clemson SC, USA

C++ Overview

- Designed by B. Stroustrup (1986),
- C++ and ANSI C
- Hybrid language: OO and 'conventional' programming,
- More than just an OO version of C
 - operator overloading
 - templates

2

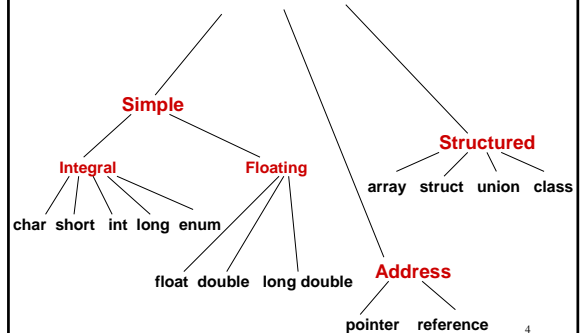
Identifiers

- An identifier must start with a letter or underscore, and be followed by zero or more letters
- **C++ is case sensitive**
- **VALID**

<code>age_of_dog</code>	<code>TaxRate98</code>
<code>PrintHeading</code>	<code>AgeOfHorse</code>

3

C++ Data Types



4

Typical size of data types

Type	Number of Bytes
char	1
short (short int)	2
int	4
unsigned int	4
enum	2
long (long int)	4
float	4
double	8
long double	12

5

Finding size of data type

- Use **sizeof** to find the size of a type
e.g.
`cout << sizeof(int)`

Find size of types
on our system.

6

Enumerated types: *enum*

- Used to define constant values

```
enum days{
    Sunday = 0, Monday, Tuesday,
    Wednesday, Thursday, Friday,
    Saturday
} yesterday, today;
```

7

Boolean type

- C++ has a built-in logical or Boolean type

```
bool flag = true;
```

8

Giving a value to a variable

In your program you can assign (give) a value to the variable by using the **assignment operator =**

```
int Age_Of_Dog = 12;
```

or by another method, such as

```
cout << "How old is your dog?";
cin >> Age_Of_Dog;
```

9

Named Constants

- A **named constant** is a location in memory which we can refer to by an identifier, and in which a **data value that cannot be changed** is stored.

VALID CONSTANT DECLARATIONS

```
const char STAR = "*";
const float PI = 3.14159;
const int MAX_SIZE = 50;
```

Note: all caps

10

some keywords: reserved to C++

- bool, true, false,
- char, float, int, unsigned, double, long
- if, else, for, while, switch, case, break,
- class, public, private, protected, new, delete, template, this, virtual,
- try, catch, throw.

frequently used keywords

11

two portable versions of main

type of returned value name of function says no parameters

```
int main ( )
{
    return 0;
}
```

the return statement is optional

main returns an integer to the operating system

12

two portable versions of main

```
int main (int argc, char * argv[] )  
{  
    return 0;  
}
```

the return statement is optional

*main returns an integer
to the operating system*

13

Modulus Operator

- The modulus operator % can only be used with integer type operands and **always has an integer type result.**
- Its result is the remainder after division

- EXAMPLE

11 % 4 is ?

$$\begin{array}{r} \text{R} = ? \\ 4 \overline{) 11} \end{array}$$

14

Prefix vs postfix

- When the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield **different** results.

```
int x = 3, y = 3;  
cout << ++x << endl;  
cout << y++ << endl;
```

what's the output?

15

Program with Three Functions

```
#include <iostream>
```

```
int Square ( int );           // declares these 2 functions  
int Cube ( int );
```

```
int main ( void ) {  
    cout << "The square of 27 is " << Square (27) << endl;  
    cout << "The cube of 27 is " << Cube (27) << endl;  
}
```

```
int Square ( int n ) { return n * n ; }  
int Cube ( int n ) { return n * n * n ; }
```

16

What's a file?

- A logical structure
- contains data in some format
 - ascii
 - binary
 - jpeg, gif
- provides logical structure to the data
- usually stored on disk

17

Operators can be

binary involving 2 operands **2 + 3**

unary involving 1 operand **- 3**

ternary involving 3 operands **?:**

18

Some C++ Operators

Precedence	Operator	Description
Higher	()	Function call
	+	Positive
	-	Negative
	*	Multiplication
	/	Division
	%	Modulus (remainder)
	+	Addition
	-	Subtraction
Lower	=	Assignment

19

Type Casting is Explicit Conversion of Type

int(4.8) has value

static_cast<float>(5)

float(2/4)

float(2) / float(4)

20

>> is a binary operator

>> is called the input or extraction operator

>> is left associative

EXPRESSION	HAS VALUE
cin >> Age	cin

STATEMENT
cin >> Age >> Weight ;

21

Some Expressions

int age;

EXAMPLE	VALUE
age = 8.5	
5 / 8	
6.0 / 5.0	
float (4 / 8)	
float (4) / 8	
cout << "How old are you?"	
cin >> age	
cout << age	

22

Parts of a Function

- Every function has 2 parts

```
int main (void)
{
    return 0;
}
```

signature

body

23

HEADER FILE	FUNCTION	EXAMPLE OF CALL	VALUE
<stdlib.h>	abs(i)	abs(-6)	6
	fabs(x)	fabs(-6.4)	6.4
<cmath>	pow(x,y)	pow(2.0,3.0)	8.0
<cmath>	sqrt(x)	sqrt(100.0)	10.0
	sqrt(x)	sqrt(2.0)	1.41421
<cmath>	log(x)	log(2.0)	.693147
<iomanip>	setprecision(n)	setprecision(3)	

24

C++ Basics

Formatted I/O

25

output of float may look like int!

```
#include <iostream>
```

```
main() {  
    float z = 99.0;  
    cout << z << endl;  
}
```

output is:
99

26

scientific notation

```
#include <iostream>  
#include <iomanip>
```

```
main() {  
    float z = 123456789.12335;  
    cout << z << endl;  
}
```

output may be:
1.23457e+08

27

Can take control of formatting

```
#include <iomanip>  
#include <iostream>  
int main ( void )  
{  
    float myNumber = 123.4 ;  
    float yourNumber = 3.14159 ;  
  
    cout.setf ( ios::fixed , ios::floatfield ) ; // use decimal format  
    cout.setf ( ios::showpoint ) ; // print decimal point  
  
    cout << "Numbers are: " << setprecision ( 4 ) << endl  
        << setw ( 10 ) << myNumber << endl  
        << setw ( 10 ) << yourNumber << endl ;  
  
    return 0 ;  
}
```

28

C++ basics

Program Input

29

Input Statements

SYNTAX:

```
cin >> Variable >> Variable ... ;
```

These examples yield the same result.

```
cin >> length ;  
cin >> width ;
```

```
cin >> length >> width ;
```

30

Whitespace Characters Include . . .

- blanks
- tabs
- end-of-line (newline) characters

The newline character is created by hitting Enter or Return at the keyboard, or by using the manipulator endl or “\n” in a program.

31

Extraction Operator >>

“skips over”

(actually reads but does not store anywhere)

leading white space characters

32

Another way to read char data

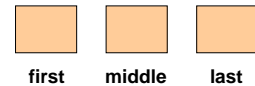
The `get()` function can be used to read a single character.

It obtains the very next character from the input stream without skipping any leading white space characters.

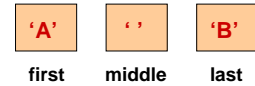
33

At keyboard you type:
A[space]B[space]C[Enter]

```
char first ;
char middle ;
char last ;
```



```
cin.get ( first ) ;
cin.get ( middle ) ;
cin.get ( last ) ;
```



NOTE: The file reading marker is left pointing to the space after the 'B' in the input stream.

34

C++ Basics

More Arithmetic Expressions,
Flow of Control
Relational & Logical
Expressions

35

Arithmetic expressions

- Assignment and addition

```
x = x + a
x += a
```

36

C++ control structures

- Selection
 - if
 - if ... else
 - switch
- Repetition
 - for loop
 - while loop
 - do ... while loop

37

CONTROL STRUCTURES

Use logical expressions which may include:

6 Relational Operators

< <= > >= == !=

3 Logical Operators

! && ||

38

Operator	Meaning	Associativity
!	NOT	Right
*, /, %	Multiplication, Division, Modulus	Left
+, -	Addition, Subtraction	Left
<	Less than	Left
<=	Less than or equal to	Left
>	Greater than	Left
>=	Greater than or equal to	Left
==	Is equal to	Left
!=	Is not equal to	Left
&&	AND	Left
	OR	Left
=	Assignment	Right

39

“SHORT-CIRCUIT” EVALUATION

- C++ uses short circuit evaluation of logical expressions
- this means that evaluation stops as soon as the final truth value can be determined

40

Short-Circuit Example

```
int Age, Height;  
Age = 25;  
Height = 70;
```

EXPRESSION

```
(Age > 50) && (Height > 60)
```

false

Evaluation can stop now

41

Better example

```
int Number;  
float X;  
(Number != 0) && (X < 1 / Number)
```

42

Compound statement

- We use braces {} to build compound - statements
- To use braces or not to use braces??

```
for (i = 0; i <= n; ++i)
    sum += i;

for (i = 0; i <= n; ++i) {
    sum += i;
}
```

43

Conditional statements

- Syntax

```
if (expression)
    statement1
else
    statement2
```

else clause is optional

44

Beware of *dangling else*



what's the output?

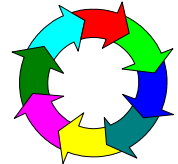
```
int x = 7, y = 8;
if (x == 0)
    if (y == 0) cout << "yes" << endl;
else cout << "no" << endl;
cout << "end of output" << endl;
```

45

Iteration statements

- while-statement syntax
- semantics

```
while (expression)
    statement
```



It's a pre-test loop.

46

Iteration statements

```
// compute sum = 1 + 2 + ... + n

int sum = 0;
i = 1;
while (i <= n) {
    sum += i;
    i++;
}
```

init of the lcv
loop termination condition.
body of the loop
incr of the lcv

47

Iteration statements

```
// compute sum = 1 + 2 + ... + n

int sum = 0;
for (int i = 1; i <= n; ++i) {
    sum += i;
}
```

i doesn't exist here!

48

Break

- `break;`
 - the execution of a loop or switch terminates immediately if, in its inner part, the `break;` statement is executed.



49

Combining *break* and *for*

```
char ch;  
int count = 0;
```

```
for (; ;) {  
    cin >> ch;  
    if (ch == '\n') break;  
    ++count;  
}
```

Alternatives?

50

Switch

```
switch (letter) {  
    case 'N': cout < "New York\n";  
              break;  
    case 'L': cout < "London\n";  
              break;  
    case 'A': cout < "Amsterdam\n";  
              break;  
    default: cout < "Somewhere else\n";  
              break;  
}
```

51

Switch

```
switch (letter) {  
    case 'N': case 'n': cout < "New York\n";  
              break;  
    case 'L': case 'l': cout < "London\n";  
              break;  
    case 'A': case 'a': cout < "Amsterdam\n";  
              break;  
    default: cout < "Somewhere else\n";  
              break;  
}
```

52

What's the output?

```
int i = 0;  
switch ( i ) {  
    case 0: i += 5;  
    case 1: ++i;  
    case 2: ++i;  
    case 3: ++i;  
    default ++i;  
}  
cout << "i is: " << i << endl;
```

53

Conditional expressions

- Ternary operator

```
expression1 ? expression2 : expression3
```

54

Conditional expression

```
// example of the conditional expression

int a = 7, b = 9;
cout << "The greater of a and b is"
    << (a > b ? a : b);
```

What's the output?

55

Simple arrays

- subscripts can be an integer expression
- In the declaration, the dimension must be a constant expression

```
const int LENGTH = 100;
...
int a[LENGTH]
...
for (int i=0; i<LENGTH; i++)
    a[i] = 0; // initialize array
```

known at compile time!

56

Type safety

- Every arithmetic expression has a type
- This type can be derived from those of its operands
- Try to be type-safe: **Don't mix types!**

57

Functions:

3 parameter transmission modes

- pass by *value* (*default*) local copy
- pass by *reference* (&) pass the address
- pass by *const reference* (*const &*)
good for big structures



58

Functions: example of pass by value

```
int sqr(int x) {
    ...
}
```

The compiler makes
a local copy!

59

Functions: example of pass by reference

```
void swap(int &x, int &y) {
    ...
}
```

Must pass by reference!

60

Functions: pass by *const reference*

- Makes sense with large structures or objects

We'll use it when
we make objects.

61

Arrays are passed by reference

```
const int MAX = 100;
void init(int a[], int x) {
    for (int i = 0; i < MAX; ++i) {
        a[i] = rand() % 100;
    }
    x = 99;
}
main() {
    int a[MAX], x = 7;
    init(a, x);
    cout << a[0] << '\t' << x << endl;
}
```

62

The void keyword

- In C one might write

```
main()
```

- This is equivalent to:

```
int main()
```

not `void main()` and implies `return 0;` at
the end of the main function.

63

Functions: Types of arguments and return values

- Types of return values
 - conversion rules also apply to return-statements

```
int g(double x, double y) {
    return x * x - y * y + 1;
}
```

- the value returned is `int` and truncation
takes place

64

Functions: Types of arguments and return values

- Types of return values
 - It would be better to explicitly acknowledge
this with a cast

```
int g(double x, double y) {
    return
        static_cast<int>(x * x - y * y + 1);
}
```

65

Functions: initialization

```
#include <iostream>

void f() {
    static int i=1;
    cout << i++ << endl;
}

int main() {
    f();
    f();
    return 0;
}
```

What's the output?

66

A static variable can be used as a flag

```
void f() {
    static bool first_time = true;
    if (first_time) {
        cout << "f called for the first time\n";
        first_time = false; // false
    }
    cout << "f called (every time)\n";
}
```

*static constant can also
be used in classes*

67

Functions: initialization

- Default arguments
 - C++ allows a function to be called with fewer arguments than there are parameters
 - Once a parameter is initialized, all subsequent parameters must also be initialized

```
void f(int i, float x=0, char ch='A') {
    ..
}
```

68

Functions: initialization

```
void f(int i, float x=0, char ch='A')
{
    ...
}
...
f(5, 1.23, 'E');
f(5, 1.23); // equivalent to f(5,1.23,'A');
f(5); // equivalent to f(5,0,'A');
```

69

Function overloading

- two or more functions with the same name
- The number or types of parameters must differ:

```
void writenum(int i) {
    cout << "i is " << i << endl;
}

void writenum(float x) {
    cout << "x is: " << x << endl;
}
```

70

Functions: overloading

```
int g(int n) {
    ...
}

float g(int n) {
    ...
}
```

Will this compile?

71

Functions: References as return values

- A value can be returned from a function using any of the 3 transmission modes.
- This is especially important when passing objects.

Pass by value makes a copy!

72

Functions: Inline functions and macros

- A function call causes
 - a jump to a separate and unique code segment
 - the passing and returning of arguments and function values
 - saving the state
- Inline functions cause
 - no jump or parameter passing
 - no state saving
 - duplication of the code segment in place of the function call



73

What's better: #define, or const

- #define is a preprocessor directive
 - thus, constants so defined are not seen by the compiler, or the debugger.
 - might get confusing error message about a number where you used a constant
- soln: use const
 - but HOW?

74

constants in a class:

```
class Game {
private:
    static const int NUM_TURNS = 5;
    int scores[NUM_TURNS];
};

const int Game::NUM_TURNS;
```

← declaration

this is ISO compliant, but some compilers might not have caught up. gcc has!

← definition

75

constants in a class (old style):

```
class Game {
private:
    enum { NUM_TURNS = 5 };
    int scores[NUM_TURNS];
};
```

affectionately known as the "enum hack"! But don't have to do it this way anymore.

76

Which is safer: macro or inline? which faster? which smaller?

```
#define max(a, b) (a < b) ? a : b

inline int max(int a, int b) { return a > b ? a : b; }

template <class T>
inline const T& max(const T& a, const T& b) {
    return a > b ? a : b;
}
```

77

Use inlining judiciously

- Inlining is safer, provides opportunity for compiler to optimize, frequently smaller and faster code!
- overzealous inlining = code bloat ==> pathological paging, reduce instruction cache hit rate
- if function body is short, inlined function may be shorter than code generated for the call
- the inline directive is a compiler hint, not a command. Compilers are free to ignore

78

Which is better printf or cout?

```
#include <iostream>

int i;
Rational r;

cout << i;
cout << r;
```

79

What's the difference between <iostream.h> and <iostream>

- <iostream> is part of the ANSI standard C++ library; it's in std namespace
- <iostream.h> is an artifact of pre-standard C++; it's not in std
- However, vendors do not want old code to break; thus, <iostream.h> will likely continue to be supported

deprecated

80

What's the difference between string.h and string

- <string.h> -- the old C-string (char *)
- <string> -- C++ string and C-string
- <cstring> -- old C-string in std namespace

81

What's the algorithm for old/new

- old headers are likely to continue to be supported, even though they're not in the ANSI standard
- old headers are deprecated
- new C++ headers have same name w/out the .h, but contents are in std
- C headers, like <stdio.h> continue, but not in std
- new C++ headers for functionality in old C library have names like <cstdio>; offer same content but are in std

82

Preprocessor facilities

- Conditional compilation
 - a useful way to 'comment out' large sections of debugging statements

```
#if DEBUG
    cout << "the value of x is: " << endl;
    ...
#endif
```

83

Preprocessor facilities

- Conditional compilation
 - a useful way to handle multiple include files

```
#ifndef SOME__HEADER__FILE
    #include "SOME__HEADER__FILE"
#endif
```

System files automatically do this:
for example <iostream.h>

84

C strings:

- C++ has them but: artifact of C
- *programmer must manage memory*
- terminated with '\0'
- built-in c-string library:
 - `strlen(s)`: returns length of string
 - `strcat(s, t)`: place *t* at end of *s*
 - `strcpy(s, t)`: copy *t* into buffer *s*
 - `strcmp(s, t)`: 0 if *s*==*t*, <0 if *s*<*t*, >0 if *s*>*t*

85

C-string exercise:

- Write a program that reads first and last name from the terminal; assume that the names are separated by a blank.
- print the name in last name, comma, first name format.
- Example:
 - read *Daniel O'Connell*
 - print *O'Connell, Daniel*

86

Command line parameters

- C++ permits the user to pass parameters from the command line
- This facility is an artifact of C

87

Command line parameters

`main(int argc, char * argv[])`
argc is the number of parameters
argv is an array of `char*` with
`argv[0]` the first parameter
`argv[1]` the second parameter
etc.

Note that there is always at least one parameter

88

Command line parameters

- Integer parameters must be converted:

```
main(int argc, char * argv[]) {
    if (argc < 2) {
        cout << "usage: " << argv[0]
            << " <number>" << endl;
        return 1;
    }
    int n = atoi(argv[1]);
```

89



Reading from files

- Files represent a permanent medium
- must be **opened**
- inside programs, files have a **logical name** that must be mapped to the **physical name** on the disk
- after reading from or writing to a file, it should be **closed**

90

To Use Disk I/O, you must

- use `#include <fstream>`
- choose valid variable identifiers for your files and declare them
- open the files and associate them with disk names
- use your variable identifiers with `>>` and `<<`
- close the files

91

Statements for using Disk I/O

```
#include <fstream>

ifstream myInfile;           // declarations
ofstream myOutfile;

myInfile.open("A:\\myIn.dat"); // open files
myOutfile.open("A:\\myOut.dat");

myInfile.close();          // close files
myOutfile.close();
```

92

What does opening a file do?

- associates the C++ identifier for your file with the physical (disk) name for the file
- if the input file does not exist on disk, open is not successful
- if the output file does not exist on disk, a new file with that name is created
- if the output file already exists, it is erased
- places a *file reading marker* at the very beginning of the file, pointing to the first character in it

93

```
#include <fstream>
main (int argc, char * argv[]) {
    fstream input;
    input.open(argv[1], ios::in);
    int count = 0;
    char ch;
    input.get(ch);
    while ( ! input.eof() ) {
        if (ch == '\n') ++count;
        input.get(ch);
    }
    cout << "there are: " << count
         << " lines in " << argv[1] << endl;
}
```

This is bad programming style

*There are no checks to see
(1) if user input the filename,
(2) does the file exist,
(3) can it be opened!*

94

```
#include <fstream>
main (int argc, char * argv[]) {
    if (argc != 2) {
        cout << "usage: " << argv[0] << " << <filename>" << endl;
        return 1;
    }
    fstream input;
    input.open(argv[1], ios::in);
    if (!input) {
        cout << "file: " << argv[1] << " does not exist!" << endl;
        return 1;
    }
    int count = 0; char ch; input.get(ch);
    while ( ! input.eof() ) {
        if (ch == '\n') ++count;    input.get(ch);
    }
    cout << "there are: " << count << " lines in " << argv[1] << endl;
}
```


Check argument count

Make sure the file exists

look for end of file

95

```
#include <fstream>
void double_space(istream & f, ostream & t) {
    char ch;
    while ( f.get(ch) ) {
        t.put(ch);
        if (ch == '\n') t.put(ch);
    }
}
int main(int argc, char * argv[]) {
    if (argc != 3) {
        cout << "usage: " << argv[0] << " << <infile> <outfile>";
        return 1;
    }
    istream fin(argv[1]);  ostream fout(argv[2]);
    if (!fin) { cout << "can't open " << argv[1] << endl; return 1; }
    if (!fout) { cout << "can't open " << argv[2] << endl; return 1; }
    double_space(fin, fout);  fin.close();  fout.close(); return 0;
}
```



96

Exercises: write a program that reads from a file and

- reads integers and counts how often a larger integer is immediately followed by a smaller one
- counts occurrences of upper case letters
- prints the sum of the digits in a sequence of chars: a2b55((c3h is 15
- writes a file with lines numbered
- converts a file from unix to DOS



97

C++ & Object-Oriented Programming



The Class



98