An Introduction to the C++ Programming Language

Jerry Ebalunode
Center for Advanced Computation and Data Systems (CACDS)
support@tlc2.uh.edu

University of Houston
Houston, TX

Overview of the Workshop

- Accessing Workshop Materials
- Writing a Basic C++ Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Control Structures
- Functions in C++
- Arrays
- Pointers
- Working with Files
- Wrap Up
  - Using C++ in Numerical Analysis
- Exercises

- **All the concepts are accompanied by examples.**
  - **Online references:**
    - http://www.cplusplus.com/reference
    - http://www.cplusplus.com/doc/tutorial
    - And Google…..

First Access Your Account

- Log into your accounts
  - Username or login = hpc_user**X**
  - Where x = sign in serial number 1 – 47
  - Password = **cacds2014**
  - Use your web browser
    - Firefox, Chromium or Google chrome
- Slides could be downloaded from URL below

  http://129.7.249.171/workshops/intro2c++.pdf

Getting Started

- Use the terminal to download intro2c++lab.zip file to your home directory
  - Run the following commands

```
cd

wget   http://129.7.249.171/workshops/intro2c++_lab.zip    ## to get tutorial package

unzip    intro2c++_lab.zip

cd  intro2c++
```

- Now, you can begin working with tutorial files on your terminal

C++ Programming Language

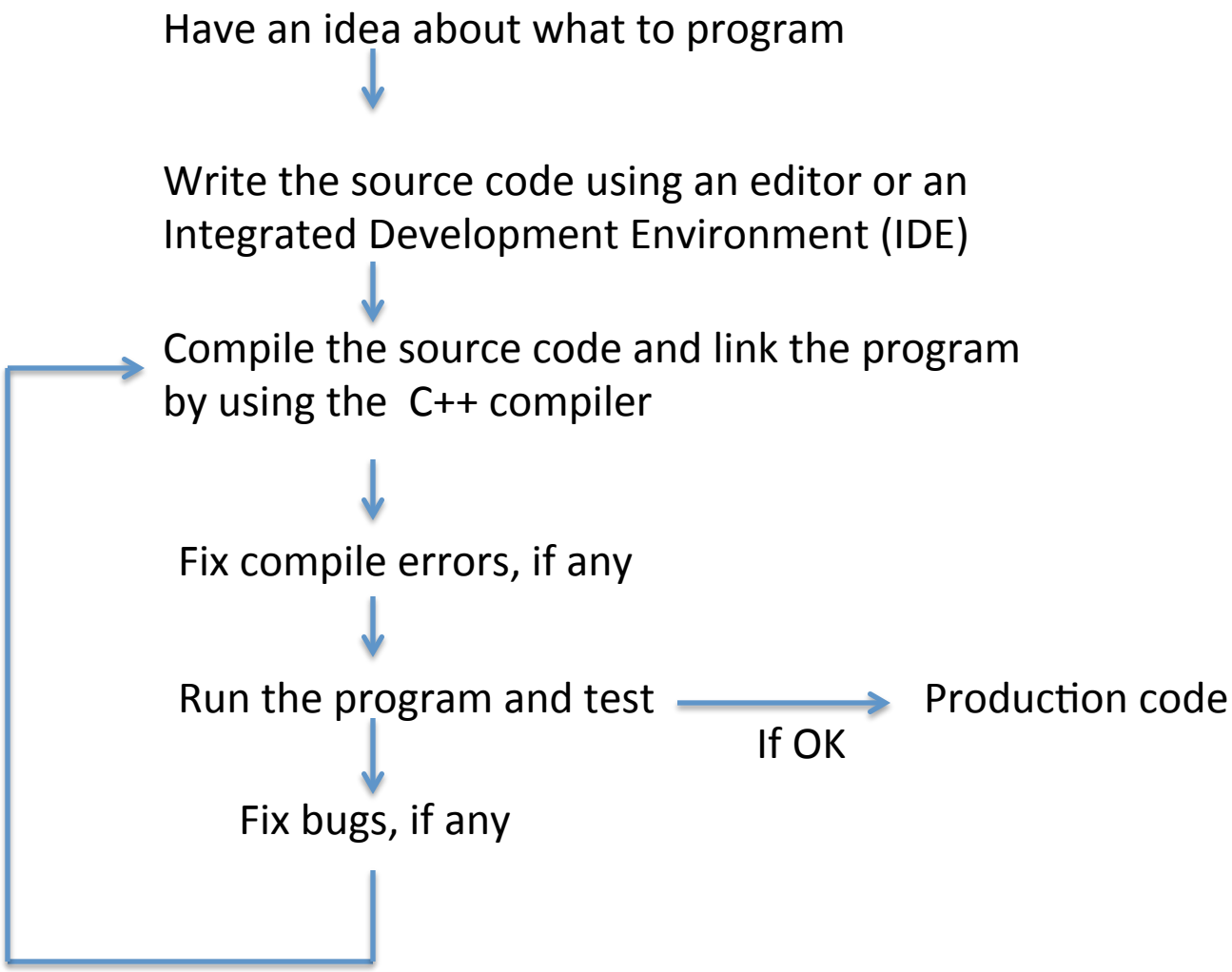
C++ is a low-level programming language

- Object-Oriented Programming
  - Classes (algorithm + data)→ driven by data and methods (data)
  - Higher level of abstraction
- Generic Programming
  - Template programming → type independence
  - Reusable code
- Procedural programming
  - Series of computational steps leading to a desired goal

It is a superset of “C” programming language

Developed by Bjarne Stroustrup

Mechanics of Creating a C++ Program



Writing the Source Code: MyFirstProgram.cpp

```
#include  <iostream>

using  namespace std ;

int      main()
{
         cout    << "Introduction to C++" << endl ;

         return 0;
}
```

A Closer Look At The Source Code: Myfirstprogram.Cpp

Preprocessor directive

Name of the standard header file to be included is specified within angular brackets

```
#include <iostream>
```

```
using namespace std ;
```

*Required for resolving **cout***

Function return type

```
int main()
```

Function name is followed by parentheses – they can be empty when no arguments are passed

```
{
```

*Output stream object for displaying information on the screen, belongs to the **namespace std**, notice the insertion operator **<<***

```
cout   << "Introduction to C++" << endl ;

return 0;
```

Keyword, command for returning function value

```
}
```

*The contents of the functions are placed inside the curly braces **{ }***

Text strings are specified within "", note <u>every statement is terminated by</u> ;

How To Create An Executable From Source Code

- Save → Compile → Link → Run
- Save your program (source code) in a file having a “cpp” extension.
  - Example, `MyFirstProgram.cpp`
    - `note C++ file extensions include`
      - **`cc ,cpp, cxx, C`**
  - Compile and Link your code (linking is done automatically by the c++ compiler)

    `c++ -o MyFirstProgram MyFirstProgram.cpp`
- Execute the program

  `./MyFirstProgram`
- Repeat the steps above every time you fix an error in code!

Different Compilers

- Different commands for different compilers (*e.g.,* **icpc** for intel compiler and **pgcpp** for pgi compiler)
  - GNU C++ compiler (**most popular and free**)

    c++ -o MyFirstCpp MyFirstCpp.cpp

  - Intel C++ compiler

    icpc -o MyFirstCpp MyFirstCpp.cpp

  - PGI C++ compiler

    pgcpp -o MyFirstCpp MyFirstCpp.cpp

- To see a list of compiler options, their syntax, and a terse explanation, execute the compiler command with the

  c++ -help or --help option

  man c++

Note we would be using “c++” compiler

Pop-Quiz 1
(add the missing components)

```
???   <iostream>
using namespace  std;
 int  main ( )
?
            cout  << "Introduction to C++“ <<  endl;

            cout  << "Enjoy the Quiz“         << endl;

             return     0;
?
```

Warnings, Errors and Bugs

- Compile-time warnings
  - Diagnostic messages
- Compile-time errors
  - Typographical errors: cuot , $include
- Link-time errors
  - Missing modules or library files
- Run-time errors
  - Null pointer assignment
- Bugs
  - Unintentional functionality

Find the Error: myError.cpp

```
#include < iostream >
using namespace std;
int main()
{
        cuot <<"Find the error"<< endl
        retrun 0;

}
```

Error Message
(compile-time error)

```
c++ -o myError myError.cpp
myError.cpp:1:22: iostream :No such
file or directory
myError.cpp: In function `int main()':
myError.cpp:6: error: `cuot' was not
declared in this scope
myError.cpp:7: error: expected `;'
before "retrun"
myError.cpp:7: error: `retrun' was not
declared in this scope
```

Comments and New Line: rules.cpp

```
/*     use  comments to describe what the code is doing
*            rules.cpp  should print one statement per line
*            this is a multi-line comment
*/
#include <iostream>
 using     namespace std;
 int main()
{
            cout << "Braces come in pairs.";
            cout << “Comments come in pairs.";
            cout << "All statements end with semicolon.";
            cout << "Every program has a main function.";

            return 0;
}
```

Output of rules.cpp

```
Braces come in pairs.Comments
come in pairs.All statements end
with a semicolon.Every program
must have  a main function.
```

Output looks odd! We want to see a new line of text for every *"cout"* statement.

Comments and New Line: rules2.cpp

```
/*
*    use  comments to describe what the code is doing
*                rules.cpp  should print one statement per line
•      this is a multi-line comment
*/

#include  <iostream>
 using namespace std;
 int main()
{
        /*notice  the  usage  of  endl
        cout   <<  “Braces     come in pairs.” << endl;
        cout  <<   “Comments come in pairs.” << endl;
        /*  \n  can  also   be    used          */
        cout  <<   “All statements end with semicolon.\n”;
        cout  <<   “Every program has a main function.” << endl;
        return 0;
}
//this  is how single line comments are specified
```

Output of rules2.cpp

```
Braces come in pairs.
Comments come in pairs.
All statements end with a semicolon.
Every program must have a main
function.
```

The output looks better now!

Variables

- Information-storage places
- Compiler makes room for them in the computer's memory
- Can contain *string, characters, numbers etc.*
- Their values can change during program execution
- All variables should be declared before they are used and should have a data type associated with them

Data Types

- Data types tell about the type of data that a variable holds
- Categories of data types are:
  - **Built-in:** `char double float long short signed unsigned void int`
  - **User-defined:** `struct union class enum`
  - **Derived:** **array or function pointer**
- We have already seen an example code in which an integer data type was used to return value from a function: `int main()`

- Compiler-dependent range of values associated with each type
  - Example: a **signed integer** can have a value in the range
  - **-32,768** to **32,767** on a 16-bit computer or
  - **-2,147,483,647** to **2,147,483,647** on a 32-bit computer
  - -2^{63} to $2^{63} - 1$ for 64 bit computer

Variable Names and Variable Declaration

- Each variable needs a name (or an identifier) that distinguishes it from other variables
- A valid variable name is a sequence of one or more alphabets, digits or underscore characters
- Keywords cannot be used as variable names
- Declaration is a statement that defines a variable
- Variable declaration includes the specification of data type and an identifier. Example:

```
int   number1;
float   number2;
```

- Multiple variables can be declared in the same statement

```
int x, y, z;
```

- Variables can be signed or unsigned
- Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values

```
signed double temperature;
```

Reading Keyboard Input: readInput1.cpp

```
#include <iostream>
using namespace std;
int main()
{
  float temperature1;
  float temperature2;
  float        average;

  cout  <<  "Enter the first temperature reading in Fahrenheit:  ";
  cin >>  temperature1;

  cout  <<  "Enter the second temperature reading in Fahrenheit:  ";
  cin >> temperature2;

  average = (temperature1 + temperature2)/2.0;
  cout  <<  "The average temperature is: " <<  average <<" F"<<  endl
  cout  <<  "The average temperature in Kelvin is: " <<  (5/9.0 * (average - 32)) + 273 <<" K"<<  endl;
  return  0;
}
```

variable declarations. It provides storage for the information you enter or **compute**.

input statement that causes the program to wait till the input is entered

```
Enter the first temperature reading in
Fahrenheit:  49
Enter the second temperature reading in
Fahrenheit:  20
The average temperature is: 34.5 F
The average temperature in Kelvin is: 274.389 K
```

Notes:

cin is the predefined object in C++ that corresponds to the standard input stream and >> operator is extraction operator

Variable Initialization

- A variable can be assigned value at the time of its declaration by using assignment operator or by constructor initialization

```
int      x   = 10;
double pi =      3.14159265358979;
char    x   =      'a';
string name =“John Doe”
```

- Variables can also be assigned values using C++ objects as in:

```
cin >> myName;
```

Scope of Variables

- A variable can be either of global or local scope
  - Global variables are defined outside all functions and they can be accessed and used by all functions in a program file
  - A local variable can be accessed only by the function in which it's created
- A local variable can be further qualified as **static**, in which case, it remains in existence rather than coming and going each time a function is called

```
static  double   pi  = 3.141592653589793;
```

- A register type of variable is placed in the machine registers for faster access – compilers can ignore this advice

```
register int x;
```

Constants and Constant Expressions

- The value of a constant never changes

```
const double   e = 2.71828182;
```

- **Useful for protecting the value of a variable like global parameters**

```
const double pi= 3.14159265358979
```

Some Operators Common in C++

Arithmetic: +, -, /, *, %, ++, --, =

Relational: a == b, a != b, a > b, a < b, a >= b, a <= b

Logical: !a, a && b, a || b

Member and Pointer: a[], *a, &a, a->b, a.b

Others: sizeof

Bitwise: ~a, a&b, a|b, a^b, a<<b, a>>b

More about operators and precedence:

http://www.cplusplus.com/doc/tutorial/operators/

Parentheses and Precedence: checkParentheses.cpp

```
#include <iostream>
using namespace std;
int main()
{
        int total;
        //multiplication has higher precedence than subtraction
        total=100-25*2;
        cout << "The total is: " << total << endl;

        //parentheses make a lot of difference!
        total=(100-25)*2;
        cout << "The total is: " << total << endl;
        return 0;
}
```

Output:

The total is: $50

The total is: $150

Overview of the Workshop

- Writing a Basic C++ Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Operators
- Control Structures
- Functions in C++
- Arrays
- Pointers
- Working with Files
- Wrap up
- Exercises

Control Structures

Selection Structure used for branching

Loop Structure used for iteration or repetition

Sequence Structure is a sequence of statements

Selection Structure
Conditional Expressions

- Using **if-else**

```
if  (a  > b)
{
    z = a;
}
else
{
    z = b;
}

// equivalent to:
z = max (a, b)
```

Loop Structures

- For repeating a sequence of steps/statements
- The statements in a loop are executed a specific number of times, or until a certain condition is met
  - Three types of loops
    - **`for`**
    - **`while`**
    - **`do-while`**

`for` Loop

```
for (start_value; end_condition; stride)
      statement;

for (start_value; end_condition; stride)
{
      statement1;
      statement2;


      statementN;
}
```

`for` Loop and break keyword
Example: forLoop.cpp

```
#include  <iostream>
using namespace std;
int main()
{
  int i;
  for ( i=0;  i<=10;  i=i+2)
  {
    if  (i >5) {  break;  }
    cout <<  "What a wonderful class!\n";
}
  return  0;
}
```

Output:
What a wonderful class!
What a wonderful class!
What a wonderful class!

`break` is the keyword used to **stop** the loop in which it is present

`while` Loop

- The while loop can be used if you don't know how many times a loop should run

```
while (condition_is_true)
{
   statement (s);
}
```

- The statements in the loop are executed till the loop condition is no longer true

- The condition that controls the loop can be modified inside the loop (this is true in the case of **for loops** too!)

`while` Loop Example: whileLoop.cpp

```
#include  <iostream>
using namespace std;
int main()
{
  int counter,  value;
  value = 5;
  counter = 0;
  while ( counter < value)
  {
    counter++;
    cout <<  "counter  value is: " <<  counter <<  endl;
  }
  return  0;
}
```

Output:

```
counter value is: 1
counter value is: 2
counter value is: 3
counter value is: 4
counter value is: 5
```

Functions in C++ Language

- Functions are self-contained blocks of statements that perform a specific task
- Written once and can be used multiple times
  - Promote code reuse
  - Makes code maintenance easy
- Two types of functions
  - Standard Library
  - User-Defined
- Like operators, C++ functions can be overloaded too

Categories of Functions

- Functions that take input and return output
- Functions that take no input, and return no output

- Functions that take input and use it but return no output
- Functions that take no input but return output

Standard Functions

- These functions are provided to the user in library files
- In order to use the functions, the user should include the appropriate library files containing the function definition
- For example, following functions are available through the math library named <cmath>
  - ceil(x)
  - cos(x)
  - exp(x)
  - log(x)
  - floor(x)
- All these functions take **double** values

Standard Function

Example: mathExample1.cpp

```
#include  <iostream>
#include  <cmath>
using namespace std;
int main()
{
        double  x = 0;
        cout  <<  "Enter  a double  value\n";
        cin >>  x;
        cout  <<"Square root  of  " <<  x <<  " is  " <<  sqrt(x);
        cout  <<  "\nLog  of  " <<  x <<  " is  " <<  log(x) <<  endl;
        return  0;
}
```

Note that the math library header is included

Standard functions available through math library

Output
Enter a double value
2.0
Square root of 2 is 1.41421
Log of 2 is 0.693147

User-Defined Function-using prototypes
Example: noInputNoReturn.cpp

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

void add();

int main()
{
     add() ;
     add() ;
    return  0;
}
void add()
{
        int a,  b,  c;
       cout  <<  "\n Enter Any 2 Numbers : ";
       cin >>  a >>  b;
       c = a + b;
       cout  <<  "\n Addition  is  : " <<  c;
}
```

Function Prototype or Declaration useful when the function is invoked before its definition is provided

Invoking the function add

Function Definition

Output:

```
Enter Any 2 Numbers : 1 2
Addition  is  : 3
Enter Any 2 Numbers : 4 5
Addition  is  : 9
```

Guidelines For Sending Input Values To Functions

- Determine the number of values to be sent to the function
- Determine the data type of the values that needs to be sent
- Declare variables having the determined data types as an argument to the function
- Use the values in the function
- Prototype the function if its definition is not going to be available before the place from where it is invoked
- Send the correct values when the function is invoked

Passing Values to Functions
Example: passValue1.cpp

```
#include  <iostream>
using namespace std;
void add( int  x, int y);
int main()
{
        int a,  b;
       cout  <<  "\n Enter Any 2 Numbers : ";
       cin >>  a >>  b;

       add ( a, b );
       return  0;
}
void add(int a, int b)
{
                int  c;
                 c= a + b;
                cout  <<  "\n Addition  is  : " <<  c <<endl;
}
```

function prototype: int ? , int ?

Actual parameters: a, b

Formal parameters: a, b

Note: The variables used as formal and actual parameters can have different names.

Passing Values to Functions from int main:

Example: passValue2.cpp

Note that the cstdlib library header is included

add(int a, int b) function returns integer (int)

Notice that main has two arguments

argc == argument count

argv == 2D array to store the arguments data

```
#include  <iostream>
#include <cstdlib>
using namespace std;
int add( int a, int b);

int main( int argc, char ** argv)
{
        int a,  b, c;
        if  (argc != 3) {
        cout  <<  "\nInsufficient num.  of  arguments.\n";
        cout  <<  "\nUsage:"  <<  argv[0] <<  " <firstNum>  <secondNum>\n";
        } else{
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        c  = add(a, b);
        cout  <<  "\n Addition  of  a and b is  : " <<  c <<endl;
        }
        return  0;
}
int add(int a, int b)
{
   return (a + b);
}
```

argv[1] holds the first number

argv[2] holds the second number

The atoi function converts the keyboard input/arguments, which is a string, into integer. It is part of the cstdlib library

Arrays

- An array is a multivariable
- It allows you to store many different values of same data type in a single unit and in a contiguous memory locations
- You can have arrays of objects as well
- Arrays are declared just like other variables, though the variable name ends with a set of square brackets

```
int myVector[3];
int myMatrix[3][3];
…
```

Arrays Example: arrayExample.cpp

```
#include  <iostream>
using namespace  std;
int main()
{
      int i;
      int age[4];
      age[0]=23;
      age[1]=34;
      age[2]=65;
      age[3]=74;
      for(i=0; i<4; i++)
      {
            cout  <<"Element: "<< i <<" Value of  age:  "<< age[i] <<"\n";
      }
       return 0;
}
```

declare an integer array containing 4 elements

Note: The number in the square brackets [] is the position number of a particular array element. The position numbers begins at 0

Output:

Element: 0 Value of age: 23

Element: 1 Value of age: 34

Element: 2 Value of age: 65

Element: 3 Value of age: 74

Pointers and Dynamic Memory Allocation

- A pointer is a variable that stores an address in memory
  - address of other variable or value

- For instance, the value of a pointer may be 42435. This number is an address in the computer's memory which is the start of some data

- C++ enables programmers to control the allocation and deallocation of memory in a program for any built-in type or user-defined type

  - This is dynamic memory management and is accomplished by the operators **new** and **delete**

  - Note: When we use fixed size arrays, static memory allocation takes place.

Managing Dynamic Memory in C++
new/delete

```
//Using new and delete  operators
int *  ip;
/*….do allocate memory to ip  …*/
 ip  = new int[3];

/*….do something with ip  …*/
ip[0]=1;
ip[1]= 2 + ip[0];
ip[2]=6;

/*….do de-allocate or free up ip  …*/
delete  [ ] ip;
```

new & delete Example: newDelete.cpp

```
#include  <iostream>
using namespace std;
int  main(){
        int numStudents,  *ptr, i , x;
        cout  <<  "Enter  the num of  students  : ";
        cin >>  numStudents;
        ptr= new int [numStudents];
        if(ptr==  NULL)
        {
                cout  <<  "\n\nMemory allocation  failed!";
                exit(1);
        }
        for (i=0; i<numStudents;  i++)
        {
                cout  <<  "\nEnter  the marks of  student_" <<  i +1  <<  " ";
                cin >> x;
                 ptr[i] =x;
        }
        for (i=0; i<numStudents;  i++)
        {
                cout  <<"student_"<<  i+1 <<" has "<< *(ptr + i);
                cout  <<  " marks\n";
        }
        delete [ ] ptr;
        return  0;
}
```

Output:

```
Enter the num of  students  : 2

Enter the marks of  student_1 21

Enter the marks of  student_2 22

student_1 has 21  marks
student_2 has 22  marks
```

File I/O

- C++ provides the following classes to perform output and input of characters to/from files:

  ofstream: Stream class to write on files

  ifstream: Stream class to read from files

  fstream: Stream class to both read and write from/to files.

- Objects of these classes are associated to a real file by opening a file as:

- open (filename, mode);

Modes of Files

- Mode is an optional parameter with a combination of the following flags

  ios::in Open for input operations

  ios::out Open for output operations

  ios::app All input operations are performed at the end of the file (i.e. append more data)

  – there are few more flags:

- More information:
- http://www.cplusplus.com/doc/tutorial/files/

Write to a file: fileWrite.cpp

```
#include  <iostream>
#include  <fstream>
using namespace std;
int main  ()
 {
    ofstream  myfile, myfile2;
    myfile.open ("example.txt");
    myfile  <<  "Writing  this  to  a file.\n";
    myfile.close();
    myfile2.open ("example.txt“,ios::app);
    myfile2  <<  “Appending  2nd line this  to same file.\n";
    myfile2.close();
    return  0;
}
```

Stream class to both read and write from/to files

Two ofstream objects created

Notice that the mode in which the file should be opened is not specified. Default mode is ios::out when ofstream object is used

file is opened under the append mode

This code creates a file called example.txt and inserts two sentences into it in the same way we are used to do with cout, but using the file stream myfile instead.

Reading From File & Writing to Console: fileReadScreenWrite.cpp

```
#include  <iostream>
#include  <fstream>
#include <string>
using namespace std;
int main () {
      string  line;
      Ifstream  myfile ("example.txt");
      if (myfile.is_open()){
            while ( myfile.good() )
            {
                getline (myfile,line);
                cout << line << endl;
      }
      myfile.close();
      }
      else
           cout  <<  "Unable to  open  file";
      return  0;
}
```

The function myfile.good() will return true in the case the stream is ready for input/ output operations, false when end of file is reached

Implementing Numerical Algorithms as C++ functions

- Case Study Numerical Integration
- Romberg Algorithm
  - The Romberg algorithm produces a triangular array of numbers, all of which are numerical estimates of the definite integral

$$\int_a^b f(x)dx$$

For example $$\pi \approx \int_0^1 \frac{4}{1+x^2}dx$$

Rombergs Method and Triangular Array for Pi

- Estimate π *with* Romberg n = 5

where n is the number of intervals

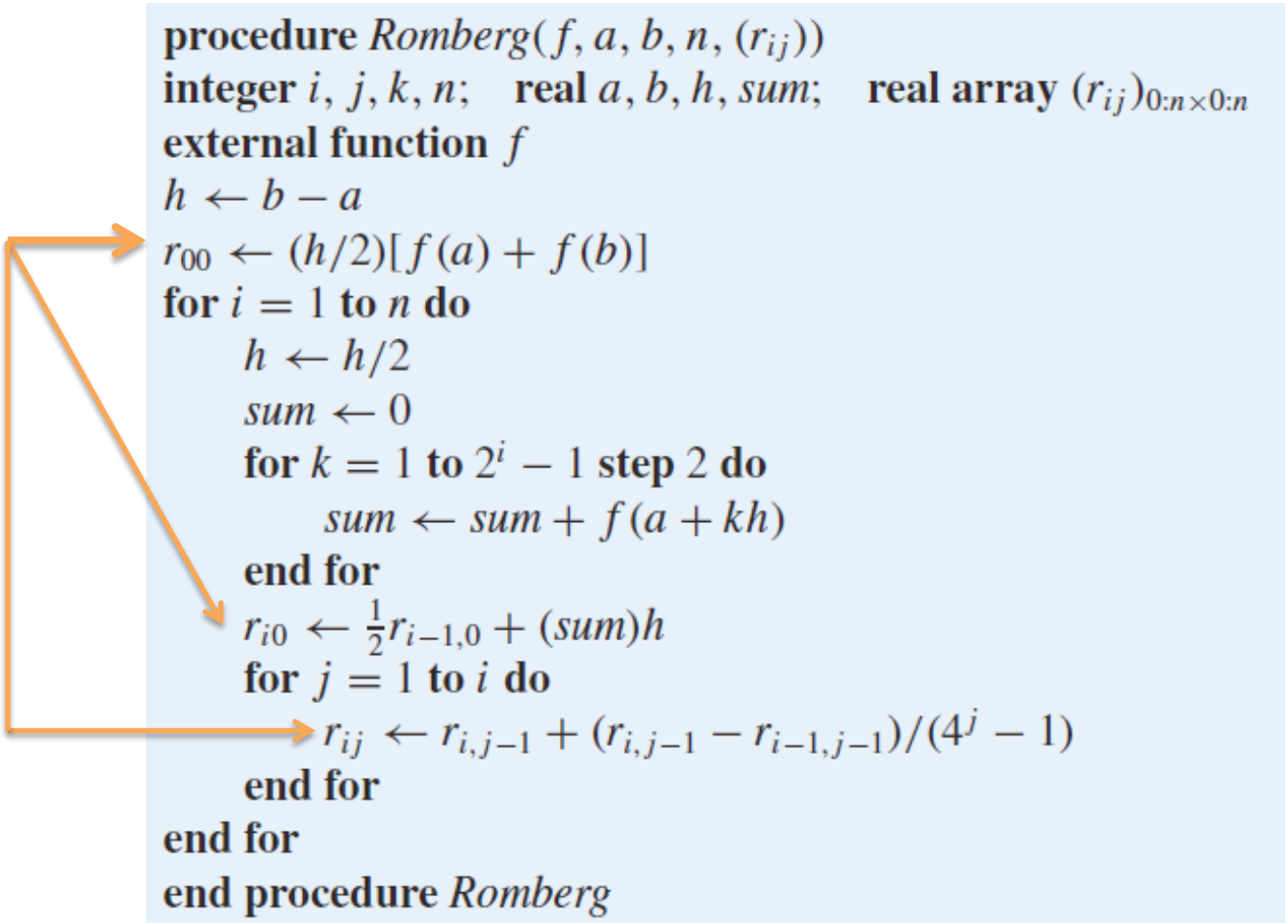
$$R(0,0) = \frac{1}{2}(b-a)(f(a)+f(b))$$

$$R(n,0) = \frac{1}{2}R(n-1,0) + h_n \sum_{k=1}^{2^{n-1}} f(a+(2k-1)h_n)$$

$$R(n,m) = R(n,m-1) + \frac{1}{4^m - 1}(R(n,m-1) - R(n-1,m-1))$$

3.00000 00000 000				
3.09999 99046 326	3.13333 32061 768			
3.13117 64717 102	3.14156 86607 361	3.14211 77387 238		
3.13898 84948 730	3.14159 25025 940	3.14159 41715 240	3.14158 58268 738	
3.14094 16198 730	3.14159 27410 126	3.14159 27410 126	3.14159 27410 126	3.14159 27410 126

Pseudo code



```
procedure Romberg(f, a, b, n, (r_ij))
integer i, j, k, n;   real a, b, h, sum;   real array (r_ij)_{0:n×0:n}
external function f
h ← b − a
r_00 ← (h/2)[f(a) + f(b)]
for i = 1 to n do
    h ← h/2
    sum ← 0
    for k = 1 to 2^i − 1 step 2 do
        sum ← sum + f(a + kh)
    end for
    r_i0 ← (1/2) r_{i−1,0} + (sum)h
    for j = 1 to i do
        r_ij ← r_{i,j−1} + (r_{i,j−1} − r_{i−1,j−1})/(4^j − 1)
    end for
end for
end procedure Romberg
```

Cheney and Kincaid, Numerical Mathematics and Computing. Sixth edition, 2008

http://en.wikipedia.org/wiki/Romberg's_method

C++ implementation: user-defined functions for estimation of Pi using Romberg's method

romberg_pi.cpp

```
#include <iostream>
#include <cmath>
using namespace std;
double f(double x)
{
  double fx;
  fx = 4.0f/(1 + x*x);
  return fx;
}

void romberg(double **r,  double a, double b, int n)
{
        int i,j,k;
         double sum, h;
         h=b-a;
         r[0][0]= h/2.0 * ( f(a) + f(b) );
        cout << r[0][0] <<endl;
        cout.precision(16);
        cout.setf(ios::fixed,ios::floatfield);
        for (i=1; i<n;i++)
        {
         h = h/2.0;
         sum=0.0;
        for (k=1; k <=  pow(2.0,i)  ;  k+=2)
         {
           sum = sum + f( a+ k*h);
         }
         r[i][0]= ( 0.5f * r[ i-1 ][0]) + sum*h;
         cout << r[i][0];
         for (j=1; j < i; j++)
         {
          r[i][j] =  r[i][j-1] + (r[i][j-1] - r[i-1][j-1])/(pow(4.0,j)-1);
          cout <<" "<<r[i][j] ;
          r[n-1][n-1]=  r[i][j] ;
         }
        cout << endl;
        }
         cout <<"Best estimate = "<< r[n-1][n-1] << endl;
}
```

C++ implementation: Main function for estimation of Pi using Romberg's method romberg_pi.cpp

```
int main (int argc , char **argv)
{
	cout << "Enter the Number of Intervals N: \n";
	int N=5, n=5;
	cin >>  N ;
	if ( N<=0) n=5;
	else n=N;
	int i;
	double a, b,  sum, **r;
	b=1.0;
	a=0.0;
	r=new double *[n];
	for (i=0;i<n; i++) r[i]=new double  [n];
	romberg(r,a,b,n);
	return 0;
}
```

Exercises

- Congrats, your team just got a 7.6% pay increase. Write a C++ program that takes in your old pay as input and prints out your new pay, taking into account your pay increase. Use a user defined function if possible.

- Use Romberg function template from the pi code to estimate (use romberg_gaussian.cpp as template)

  the Gaussian function that is integrated from 0 to 1, i.e. the error function

  – erf(1) ≈ 0.842700792949715

  – $$erf(1) \approx \frac{2}{\sqrt{\pi}} \int_0^1 e^{-t^2} dt$$

References

- http://cplusplus.com
- “Professional C++” by Gregorie, Solter and Kleper
- Numerical Mathematics and Computing by Cheney and Kincaid

Acknowledgements

- Center for Advanced Computation and Data Services
  - Barbara Chapman
  - Julie Germain
  - Manuel, & Steve
- XSEDE/TERAGRID TACC user training team
  - Training materials

If-else: Logical Expressions

```
if (temp > 75  && temp < 80)
{
  cout  <<  "It's nice  weather outside\n";
}

if  (value  ==  'e' ||  value ==  'n' )
{
cout  <<  "Exiting  the program.\n";
}
else
{
cout  <<  "\nIn the program.\n";
}
```

Decision Making, Multi-Way Decisions

- Decisions are expressed by **`if-else`** where the **`else`** part is optional

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

- Multi-way decisions are expressed using else-if statements

```
if (expression1)
      statement1
else if (expression2)
           statement2
else
         statement3
```

More on Pointers

- A pointer is a variable that stores an address in memory
  - address of other variable or value
- For instance, the value of a pointer may be 42435. This number is an address in the computer's memory which is the start of some data
- We can dereference the pointer to look at or change the data
- Just like variables, you have to declare pointers before you use them
- The data type specified with pointer declaration is the data type of the variable the pointer will point to

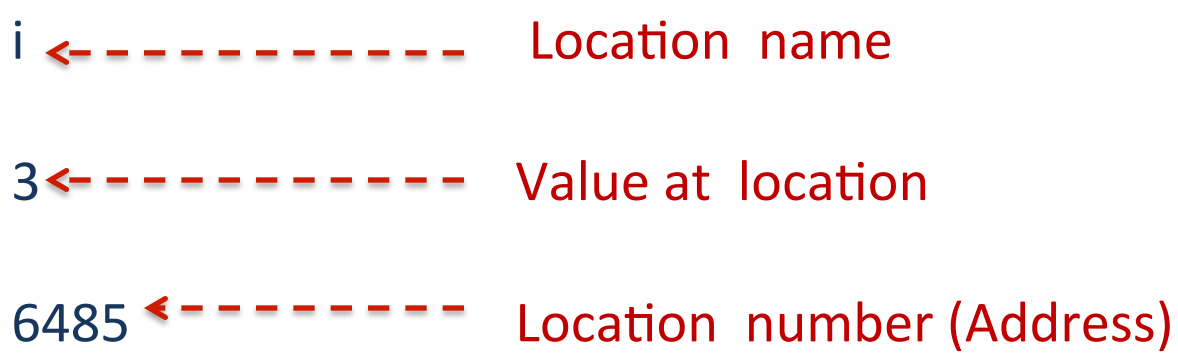
Revisiting Variable Declaration

- Consider the declaration

```
int i = 3;
```

This declaration tells the C++ compiler to:

- Reserve space in memory to hold the integer value
- Associate the name **i** with this memory location
- Store the value **3** at this location



Pointers Example: ptrExample.cpp

```
#include  <iostream>
using  namespace std;
int  main()
{
     int myValue;
     int *myPtr;
     myValue = 15;
     myPtr = &myValue;
     cout  <<  "myValue  is  equal to  " <<  myValue <<endl;
     *myPtr  = 25;
     cout  <<  "myValue  is  equal to  : " <<  myValue <<endl;
}
```

```
Output:
myValue is  equal  to  :
15
myValue is  equal to  : 25
```

Pointers and Arrays

- The square-bracket array notation is a short cut to prevent you from having to do pointer arithmetic

```
char  array[5];

array[2]  = 12;

array is a pointer to array[0]
```

array[2] = 12; is therefore equivalent to

```
*(array+2)  = 12;
```