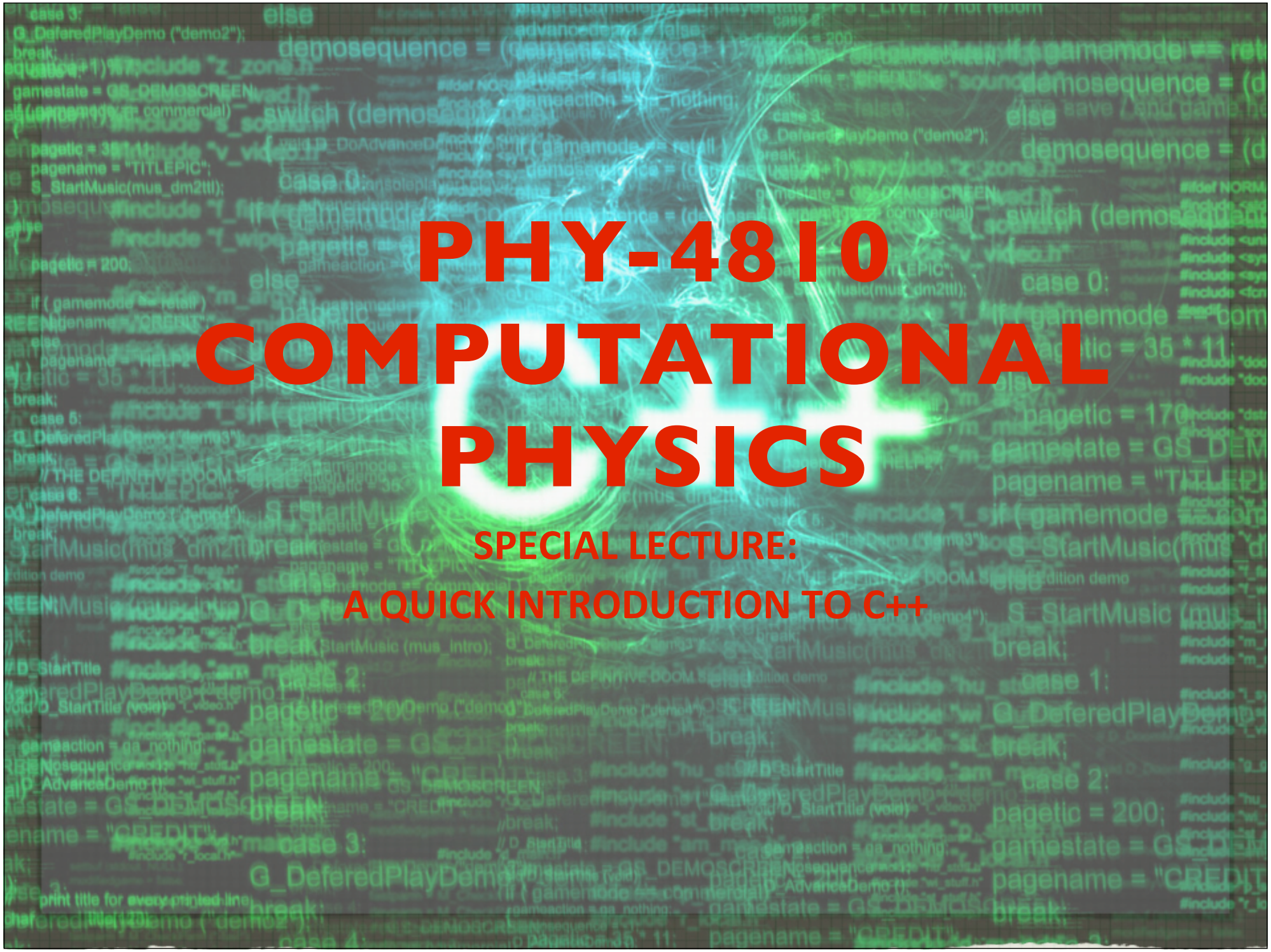
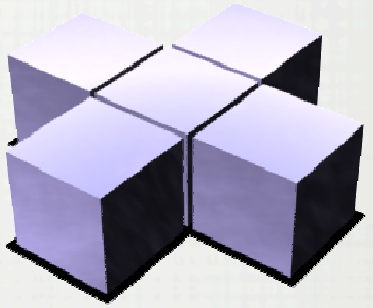


PHY-4810 COMPUTATIONAL PHYSICS

SPECIAL LECTURE:

A QUICK INTRODUCTION TO C++

ON THE MENU TODAY



- **Basics of C++**
  - **No special structures covered**
  - **No pointers covered**
  - **No functions covered**

- **Main reference: http://www.cplusplus.com/files/tutorial.pdf.**

PROGRAM STRUCTURE: FIRST PROGRAM

```
/ my first program in C++

include <iostream>
sing namespace std;

nt main ()

 cout << "Hello World!";
 return 0;
```

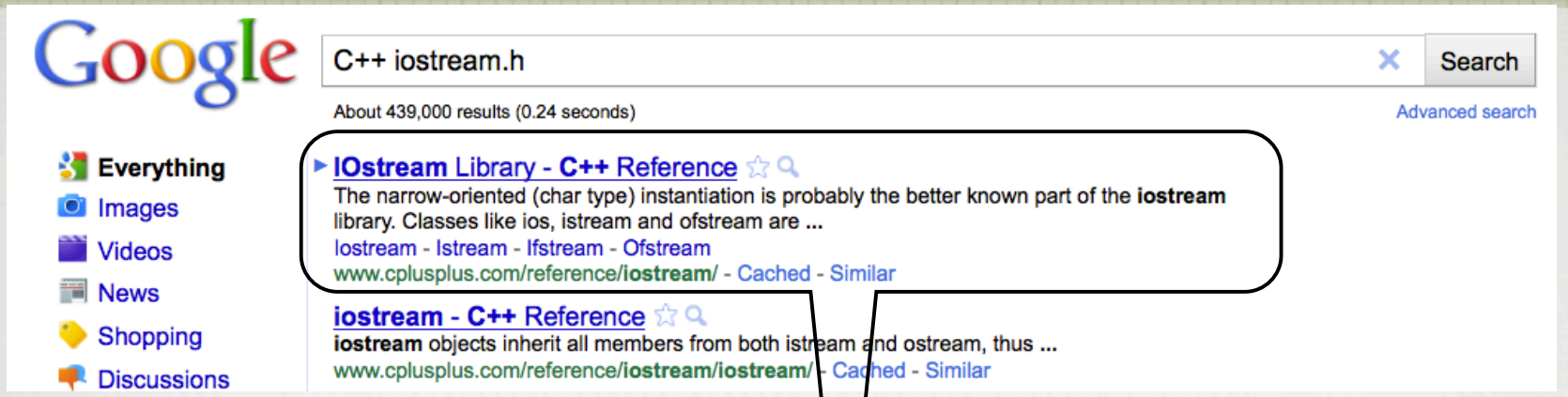
```
Hello World!
```

PROGRAM STRUCTURE: FIRST PROGRAM

```
/ my first program in C++

include <iostream>
sing namespace std;

nt main ()

 cout << "Hello World!";
 return 0;
```

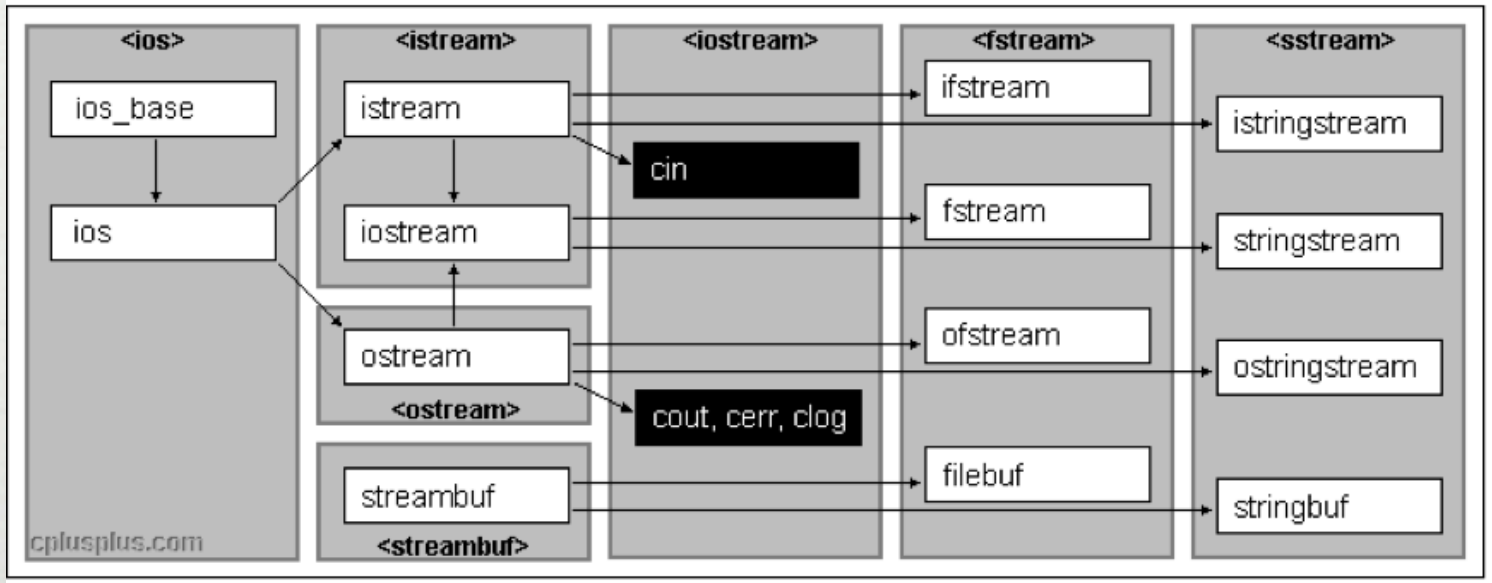
```
Hello World!
```

- // my first program in C++
  - This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program.
- #include <iostream>
  - Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the



IOstream Library

Standard Input / Output Streams Library



PROGRAM STRUCTURE: FIRST PROGRAM

```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

```
Hello World!
```

- **using namespace std;**
  - **All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name std.**
  - **So in order to access its functionality we declare with this expression that we will be using these entities.**
  - **At this point, use this line and do not worry too much about it.**

PROGRAM STRUCTURE: FIRST PROGRAM

```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

```
Hello World!
```

- **int main ()**
  - **This line corresponds to the beginning of the definition of the main function.**
  - **The main function is the point by where all C++ programs start their execution**
  - **The word main is followed in the code by a pair of parentheses (()).**
  - **In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name.**
  - **Optionally, these parentheses may enclose a list of parameters within them.**
  - **Right after these parentheses we can find the body of the main function enclosed in braces ({}).**

PROGRAM STRUCTURE: FIRST PROGRAM

```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

```
Hello World!
```

- **cout << "Hello World!";**
  - **This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.**
  - **cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters.**
  - **cout is declared in the iostream standard file within the std namespace.**
  - **Notice that the statement ends with a semicolon character (;).**
  - **(one of the most common syntax errors is indeed to forget to include some semicolon after a statement).**

PROGRAM STRUCTURE: FIRST PROGRAM

```
// my first program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World!";
  return 0;
}
```

```
Hello World!
```

- **return 0;**
  - **The return statement causes the main function to finish.**
  - **return may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution.**
  - **This is the most usual way to end a C++ console program.**

SECOND PROGRAM

```
// my second program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

SECOND PROGRAM

```
// my second program in C++

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

```
Hello World! I'm a C++ program
```

COMMENT ON COMMENTS (PAGE 10)

```
// line comment
/* block comment */
```

VARIABLES (PAGE 11): EXAMPLE OF A SIMPLE CALCULATION

```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

- **Each variable needs an identifier that distinguishes it from the others, for example, in the previous code the variable identifiers were a, b and result, but we could have called the variables any names we wanted to invent, as long as they were valid identifiers.**
- **These need to be declared in order to tell the compiler what (and how) memory space to use for them**

VARIABLES: IDENTIFIERS

- A valid identifier is a sequence of one or more letters, digits or underscore characters (_).
- Neither spaces nor punctuation marks or symbols can be part of an identifier.
  - Only letters, digits and single underscore characters are valid.
  - Variable identifiers always have to begin with a letter.
  - They can also begin with an underline character (_), but in some cases these may be reserved for compiler specific keywords or external identifiers, as well as identifiers containing two successive underscore characters anywhere. In no case they can begin with a digit.
- You cannot use a C++ keyword
- CASE SENSITIVE

```
asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete,
do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto,
if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register,
reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template,
this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void,
volatile, wchar_t, while
```

VARIABLES: DATA TYPES

Name	Description	Size*	Range*
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int` (`short`)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int` (`long`)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1byte	true or false
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`wchar_t`	Wide character.	2 *or* 4 bytes	1 wide character

VARIABLES: DECLARATION

- **Examples of valid declarations:**

```
int a;
float mynumber;
```

```
int a, b, c;
```

- **Equivalent to:**

```
int a;
int b;
int c;
```

PUTTING TO ALL TOGETHER

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

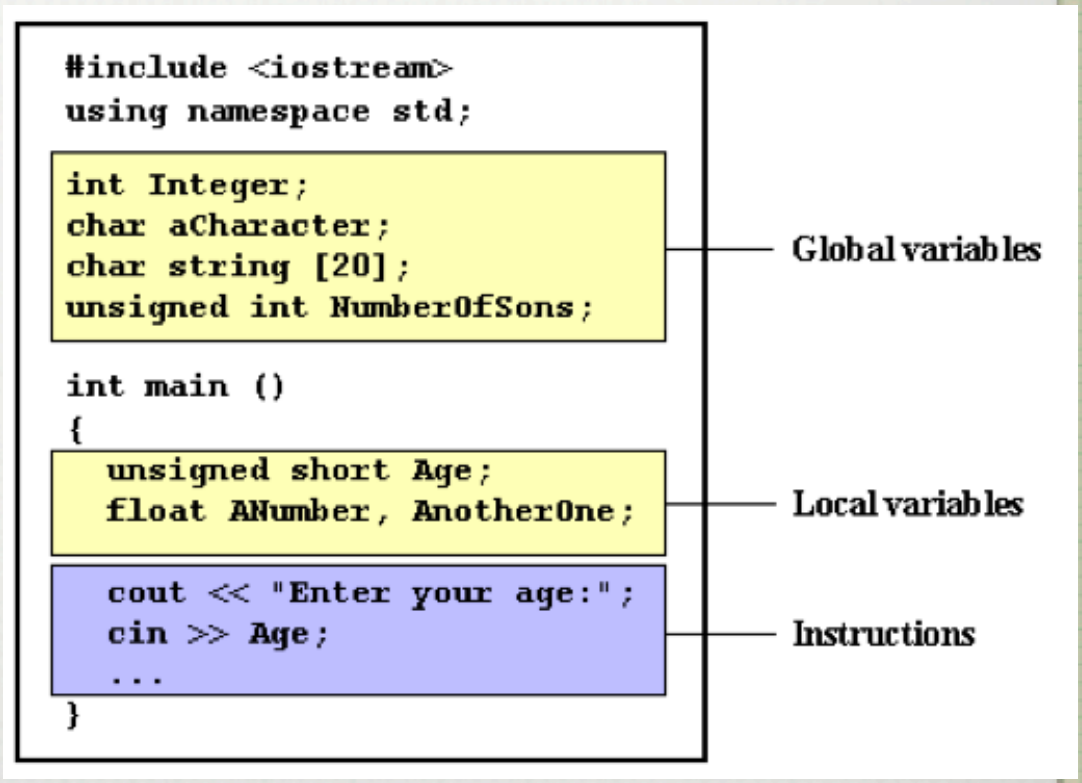
PUTTING TO ALL TOGETHER

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

```
4
```

SCOPE OF VARIABLES*

- All the variables that we intend to use in a program must have been declared with its type specifier in an earlier point in the code.
- A variable can be either of **global** or **local** scope.
  - A global variable is a variable declared in the main body of the source code, outside all functions;
  - A local variable is one declared within the body of a function or a block.



VARIABLE INITIALIZATION

- When declaring a regular local variable, its value is by default undetermined.
- But you may want a variable to store a concrete value at the same moment that it is declared. In order to do that, you can initialize the variable. There are two ways to do this in C++:
  - initialization at same time as declaration:

```
int a = 0;
```

  - "constructor initialization":

```
int a (0);
```

INTRODUCTION TO STRINGS

- **See pages 15-16**

CONSTANTS

- **Defined constant:**

```
#define PI 3.14159
#define NEWLINE '\n'
```

- **Declared constant:**

```
const int pathwidth = 100;
const char tabulator = '\t';
```

OPERATORS

- **Assignment: "="**
- **Arithmetic operators: "+,-,*,/,%"** **(modulo:** `a = 11 % 3;` **)**
- **Compound assignments: "+=, -=, *=, /=, %=, >>=, <<=, &=,^=, |="**

expression	is equivalent to
`value += increase;`	`value = value + increase;`
`a -= 5;`	`a = a - 5;`
`a /= b;`	`a = a / b;`
`price *= units + 1;`	`price = price * (units + 1);`

- **Increase and decrease (++,--)**

Example 1	Example 2

`B=3;`
`A=++B;`
`// A contains 4, B contains 4`
	`B=3;`
	`A=B++;`
	`// A contains 3, B contains 4`

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)
(5 > 4)
(3 != 2)
(6 >= 6)
(5 < 5)
```

```
(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)
(3 != 2)
(6 >= 6)
(5 < 5)
```

```
(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)
(6 >= 6)
(5 < 5)
```

```
(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)
(5 < 5)
```

```
(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)      // evaluates to true.
(5 < 5)
```

```
(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)      // evaluates to true.
(5 < 5)       // evaluates to false.
```

```
(a == 5)
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)      // evaluates to true.
(5 < 5)       // evaluates to false.
```

```
(a == 5)      // evaluates to false since a is not equal to 5.
(a*b >= c)
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)      // evaluates to true.
(5 < 5)       // evaluates to false.
```

```
(a == 5)      // evaluates to false since a is not equal to 5.
(a*b >= c)    // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c)
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)      // evaluates to true.
(5 < 5)       // evaluates to false.
```

```
(a == 5)      // evaluates to false since a is not equal to 5.
(a*b >= c)    // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c)   // evaluates to false since (3+4 > 2*6) is false.
((b=2) == a)
```

OPERATORS (CONT'D)

- **Relational and equality operators (==, !=, >, <, >=, <=)**

==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

```
(7 == 5)      // evaluates to false.
(5 > 4)       // evaluates to true.
(3 != 2)      // evaluates to true.
(6 >= 6)      // evaluates to true.
(5 < 5)       // evaluates to false.
```

```
(a == 5)      // evaluates to false since a is not equal to 5.
(a*b >= c)    // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c)   // evaluates to false since (3+4 > 2*6) is false.
((b=2) == a) // evaluates to true.
```

OPERATORS (CONT'D)

- **Logical operators (!, &&, ||)**
  - **! performs operation NOT**
  - **&& operator: AND**

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

  - **|| operator: OR**

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

MORE OPERATORS

- **Conditional operators (?)**

```
condition ? result1 : result2
```

```
7==5 ? 4 : 3      // returns 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3    // returns 4, since 7 is equal to 5+2.
5>3 ? a : b       // returns the value of a, since 5 is greater than 3.
a>b ? a : b       // returns whichever is greater, a or b.
```

- **Example:**

```
// conditional operator

#include <iostream>
using namespace std;

int main ()
{
  int a,b,c;

  a=2;
  b=7;
  c = (a>b) ? a : b;

  cout << c;

  return 0;
}
```

MORE OPERATORS

- **Conditional operators (?)**

```
condition ? result1 : result2
```

```
7==5 ? 4 : 3      // returns 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3    // returns 4, since 7 is equal to 5+2.
5>3 ? a : b       // returns the value of a, since 5 is greater than 3.
a>b ? a : b       // returns whichever is greater, a or b.
```

- **Example:**

```
// conditional operator

#include <iostream>
using namespace std;

int main ()
{
  int a,b,c;

  a=2;
  b=7;
  c = (a>b) ? a : b;

  cout << c;

  return 0;
}
```

```
7
```

EVEN MORE ON OPERATORS

- **Explicit type casting operator**

```
int i;
float f = 3.14;
i = (int) f;
```

- **sizeof()**
  - **returns the size in bytes of that type of object**

PRECEDENCE OF OPERATORS

- **when in doubt... add parentheses**

```
a = 5 + 7 % 2
```

- **Could be one of these two:**

```
a = 5 + (7 % 2)     // with a result of 6, or
a = (5 + 7) % 2     // with a result of 0
```

-

PRECEDENCE OF OPERATORS

- **when in doubt... add parentheses**

```
a = 5 + 7 % 2
```

- **Could be one of these two:**

```
a = 5 + (7 % 2)     // with a result of 6, or
a = (5 + 7) % 2     // with a result of 0
```

- **Correct answer is the first one**

BASIC INPUT-OUTPUT

- **Output**
  - **cout**
  - **insertion operator <<**

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the content of x on screen
```

```
cout << "Hello, " << "I am " << "a C++ statement";
```

```
cout << "Hello, I am " << age << " years old and my zipcode is " << zipcode;
```

IT IS IMPORTANT TO NOTICE THAT COUT DOES NOT ADD A LINE BREAK AFTER ITS OUTPUT UNLESS WE EXPLICITLY INDICATE IT, THEREFORE, THE FOLLOWING STATEMENTS:

```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

SOLUTION (1):

```
cout << "First sentence.\n ";
cout << "Second sentence.\nThird sentence.";
```

SOLUTION (2):

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

BASIC INPUT-OUTPUT

- **Input**
  - **cin**
  - **extraction operator >>**

```
int age;
cin >> age;
```

  - **Note:** `cin >> a >> b;`

```
cin >> a;
cin >> b;
```

  - **cin stops reading once it finds a blank character!!!!!!**

BASIC INPUT-OUTPUT

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

BASIC INPUT-OUTPUT

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

```
Please enter an integer value: 702
The value you entered is 702 and its double is
1404.
```

STRINGS

- **see pages 31, 32**

CONTROL STRUCTURES

- **Conditional structure: *if...else***
- **Iteration structures (loops)**
  - ***while* loop**
  - ***do-while* loop**
  - ***for* loop**
- **Jump statements**
  - ***break***
  - ***continue***
  - ***goto***
  - ***exit***
- **Switch** (case)

`if (condition) statement`

CONDITIONAL STRUCTURE

```
if (x == 100)
  cout << "x is 100";
```

SIMPLEST FORM

```
if (x == 100)
{
   cout << "x is ";
   cout << x;
}
```

USE {} FOR MORE THAN ONE INSTRUCTIONS

```
if (x == 100)
  cout << "x is 100";
else
  cout << "x is not 100";
```

ELSE

```
if (x > 0)
  cout << "x is positive";
else if (x < 0)
  cout << "x is negative";
else
  cout << "x is 0";
```

ELSE IF

```
while (expression) statement
```

ITERATIONS (LOOPS): WHILE LOOP

```
// custom countdown using while

#include <iostream>
using namespace std;

int main ()
{
  int n;
  cout << "Enter the starting number > ";
  cin >> n;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "FIRE!\n";
  return 0;
}
```

MAKE SURE THE LOOP EVENTUALLY STOPS!

```
while (expression) statement
```

ITERATIONS (LOOPS): WHILE LOOP

```
// custom countdown using while

#include <iostream>
using namespace std;

int main ()
{
  int n;
  cout << "Enter the starting number > ";
  cin >> n;

  while (n>0) {
    cout << n << ", ";
    --n;
  }

  cout << "FIRE!\n";
  return 0;
}
```

```
Enter the starting number > 8
8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

MAKE SURE THE LOOP EVENTUALLY STOPS!

```
do statement while (condition);
```

ITERATIONS (LOOPS): DO-WHILE LOOP

```
// number echoer

#include <iostream>
using namespace std;

int main ()
{
  unsigned long n;
  do {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " << n << "\n";
  } while (n != 0);
  return 0;
}
```

THE LOOP IS ALWAYS PERFORMED AT LEAST ONCE

```
do statement while (condition);
```

ITERATIONS (LOOPS): DO-WHILE LOOP

```
// number echoer

#include <iostream>
using namespace std;

int main ()
{
  unsigned long n;
  do {
    cout << "Enter number (0 to end): ";
    cin >> n;
    cout << "You entered: " << n << "\n";
  } while (n != 0);
  return 0;
}
```

```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

THE LOOP IS ALWAYS PERFORMED AT LEAST ONCE

```
for (initialization; condition; increase) statement;
```

ITERATION: FOR LOOP

```
// countdown using a for loop
#include <iostream>
using namespace std;
int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

- The initialization and increase fields are optional.
- They can remain empty, but in all cases the semicolon signs must be written.
- For example we could write:
  - for (;n<10;) if we wanted to specify no initialization and no increase;
  - for (;n<10;n++) if we wanted to include an increase field but no initialization
- **Common mistake: use == or = as appropriate**

```
for (initialization; condition; increase) statement;
```

ITERATION: FOR LOOP

```
// countdown using a for loop
#include <iostream>
using namespace std;
int main ()
{
  for (int n=10; n>0; n--) {
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, FIRE!
```

- The initialization and increase fields are optional.
- They can remain empty, but in all cases the semicolon signs must be written.
- For example we could write:
  - for (;n<10;) if we wanted to specify no initialization and no increase;
  - for (;n<10;n++) if we wanted to include an increase field but no initialization
- **Common mistake: use == or = as appropriate**

JUMP STATEMENTS

- **break:** leave the loop immediately
- **continue:** skip the rest of the loop, continue to next iteration
- goto: jump from one step to another one using label (identifier followed by a colon :)
  - never use it

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

JUMP STATEMENTS

- **break:** leave the loop immediately
- **continue:** skip the rest of the loop, continue to next iteration
- goto: jump from one step to another one using label (identifier followed by a colon :)
  - never use it

```
// continue loop example
#include <iostream>
using namespace std;

int main ()
{
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

```
10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!
```

EXIT FUNCTION

- **The purpose of exit is to terminate the current program with a specific exit code.**
- **Its prototype is:**

```
void exit (int exitcode);
```

(SELECTIVE STRUCTURE: SWITCH)

```
switch (expression)
{
  case constant1:
     group of statements 1;
     break;
  case constant2:
     group of statements 2;
     break;
  .
  .
  .
  default:
     default group of statements
}
```

(SELECTIVE STRUCTURE: SWITCH)

```
switch (expression)
{
  case constant1:
     group of statements 1;
     break;
  case constant2:
     group of statements 2;
     break;
  .
  .
  .
  default:
     default group of statements
}
```

SWITCH VERSUS IF

switch example	if-else equivalent

`switch (x) {`
`  case 1:`
`  cout << "x is 1";`
`  break;`
`  case 2:`
`  cout << "x is 2";`
`  break;`
`  default:`
`  cout << "value of x unknown";`
`  }`
	`if (x == 1) {`
	`  cout << "x is 1";`
	`  }`
	`else if (x == 2) {`
	`  cout << "x is 2";`
	`  }`
	`else {`
	`  cout << "value of x unknown";`
	`  }`

(SELECTIVE STRUCTURE: SWITCH)

```
switch (expression)
{
  case constant1:
     group of statements 1;
     break;
  case constant2:
     group of statements 2;
     break;
  .
  .
  .
  default:
     default group of statements
}
```

SWITCH VERSUS IF

switch example	if-else equivalent

switch (x) {
  case 1:
  cout << "x is 1";
  break;
  case 2:
  cout << "x is 2";
  break;
  default:
  cout << "value of x unknown";
  }
	if (x == 1) {
	  cout << "x is 1";
	  }
	else if (x == 2) {
	  cout << "x is 2";
	  }
	else {
	  cout << "value of x unknown";
	  }

BE CAREFUL:

```
switch (x) {
  case 1:
  case 2:
  case 3:
    cout << "x is 1, 2 or 3";
    break;
  default:
    cout << "x is not 1, 2 nor 3";
  }
```

FUNCTIONS*

- ☐ **pp 41 to 53**

ARRAYS

- An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.
- That means that, for example, we can store 5 values of type int in an array without having to declare 5 different variables, each one with a different identifier. Instead of that, using an array we can store 5 different values of the same type, int for example, with a unique identifier.
- Like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

```
int billy [5];
```

- we can store 5 elements in “billy”
  - STARTING at billy[0] to billy[4]

ARRAYS: INITIALIZATION

- **First method:** `int billy [5] = { 16, 2, 77, 40, 12071 };`
- **Second method:** `int billy [] = { 16, 2, 77, 40, 12071 };`
  - **in this case the compiler finds out the size automatically**

ARRAYS: ACCESSING VALUES OF ARRAY

- **Storing:** `billy[2] = 75;`
- **Using:** `a = billy[2];` **(third element of billy)**

```
billy[0] = a;
billy[a] = 75;
b = billy [a+2];
billy[billy[a]] = billy[2] + 5;
```

- **At this point it is important to be able to clearly distinguish between the two uses that brackets[ ] have related to arrays. They perform two different tasks:**
  - **one is to specify the size of arrays when they are declared; and**
  - **the second one is to specify indices for concrete array elements.**
- **Do not confuse these two possible uses of brackets [ ] with arrays.**

ARRAY: EXAMPLE

```
// arrays example
#include <iostream>
using namespace std;

int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += billy[n];
  }
  cout << result;
  return 0;
}
```

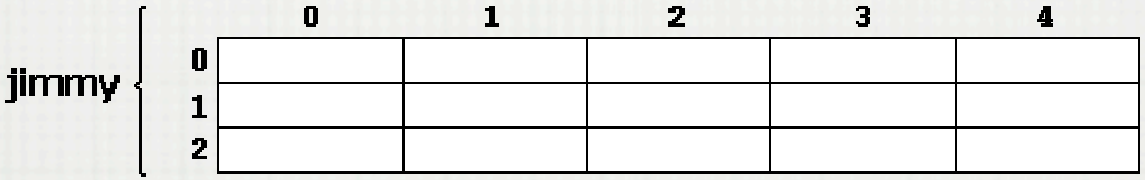
ARRAY: EXAMPLE

```
// arrays example
#include <iostream>
using namespace std;

int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;

int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += billy[n];
  }
  cout << result;
  return 0;
}
```

```
12206
```

MULTIDIMENSIONAL ARRAYS

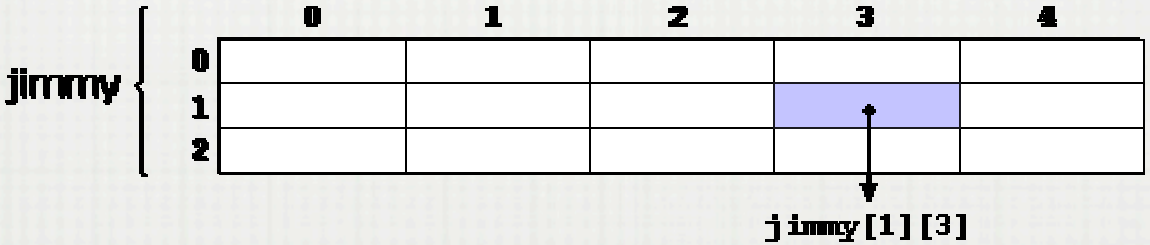
```
int jimmy [3][5];
```



- jimmy [1][3]?

MULTIDIMENSIONAL ARRAYS

```
int jimmy [3][5];
```



- jimmy [1][3]?

(POINTERS)

(DYNAMIC MEMORY)

(DATA STRUCTURE)

(CLASSES)

WRITING TO A FILE

```
#include <iostream>
#include <fstream.h>

using namespace std;
int main () {
    // insert code here...
  int i, n=100;
  ofstream FO;

  FO.open("output.dat");
  for (i=0;i<n;i++){
     FO << i << " " << i*i << endl;

  }
  FO.close();
}
```

HOW TO COMPUTE MACHINE PRECISION?

- **SINGLE PRECISION**

- **float x,epsilon;**

```
x=1;
do  {
    epsilon=x;
    x/=2;
} while (x+1>1);
cout << "Epsilon=" << epsilon <<
endl;
```

- **Epsilon=1.19209e-07**

- **DOUBLE PRECISION**

- **double x,epsilon;**

```
x=1;
do  {
    epsilon=x;
    x/=2;
} while (x+1>1);
cout << "Epsilon=" << epsilon <<
endl;
```

- **Epsilon=2.22045e-16**

```cpp
#include <iostream>
using namespace std;
//find machine precision

int main () {
   float x,epsilon;

//while
   x=1;
   while (x+1>1) {
       x/=2;
   };
   epsilon=x*2;
   cout << "Epsilon while =\t" << epsilon << endl;

//dowhile
   x=1;
   do  {
       epsilon=x;
       x/=2;
   } while (x+1>1);
   cout << "Epsilon DW =\t" << epsilon << endl;

//for
   x=1;
   for(;1;)  {
       x/=2;
       if(x+1==1) break;
   }
   epsilon=x*2;
   cout << "Epsilon for =\t" << epsilon << endl;
//
    return 0;
}
```

SINGLE PRECISION

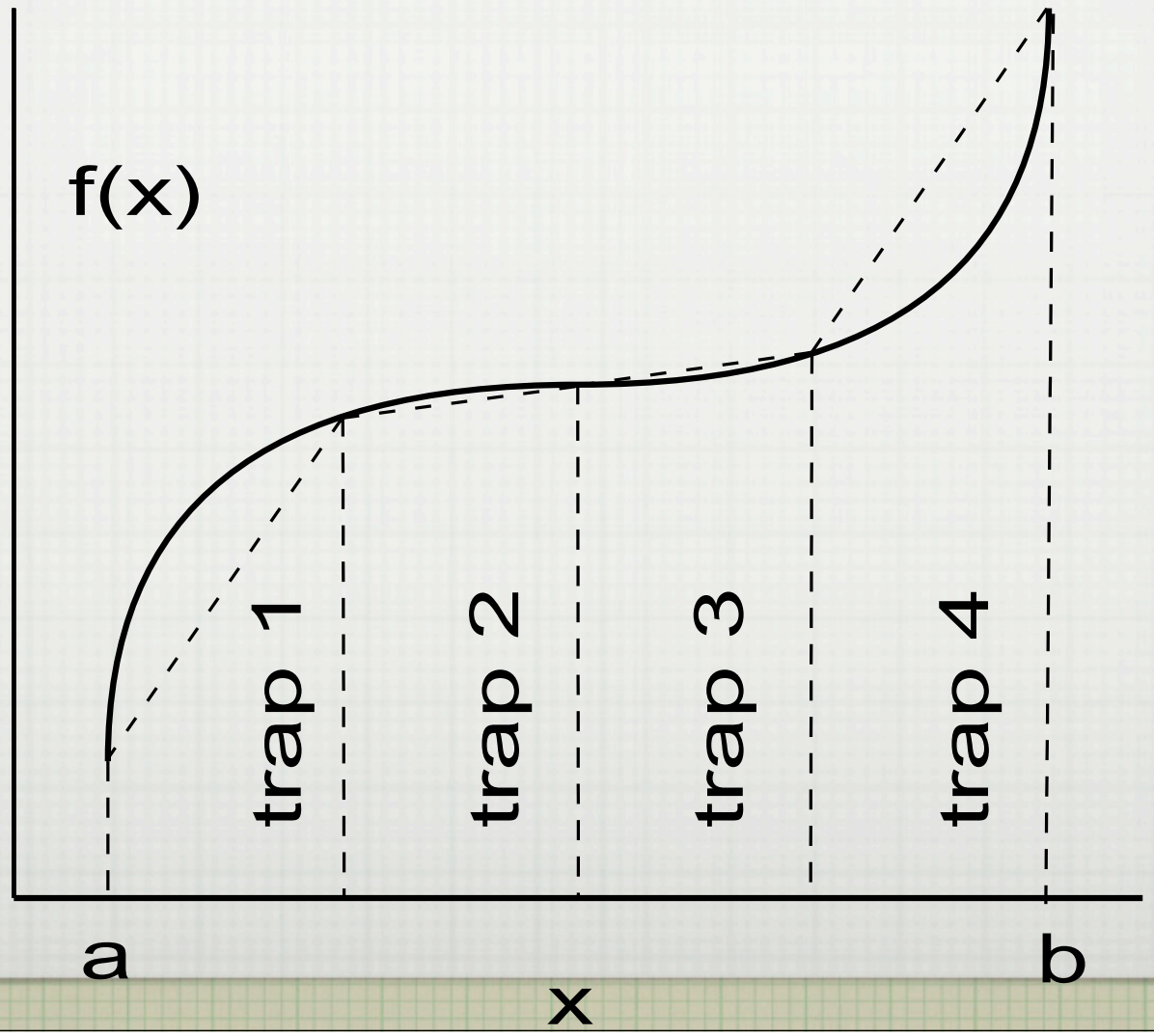
```
vinces-MacBook-Pro:Debug vm2$ ./machineprecision
Epsilon while = 1.19209e-07
Epsilon DW =    1.19209e-07
Epsilon for =   1.19209e-07
vinces-MacBook-Pro:Debug vm2$
```

DOUBLE PRECISION

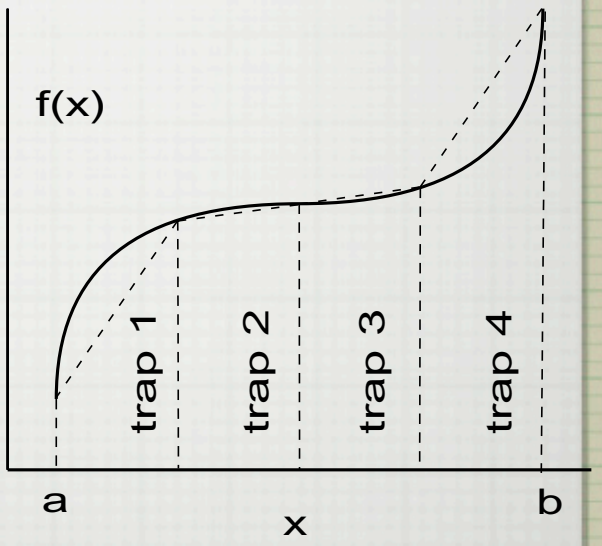
```cpp
#include <iostream>
using namespace std;
//find machine precision

int main () {
   double x,epsilon;

//while
   x=1;
   while (x+1>1) {
        x/=2;
   };
   epsilon=x*2;
   cout << "Epsilon while =\t" << epsilon << endl;

//dowhile
   x=1;
   do  {
        epsilon=x;
        x/=2;
   } while (x+1>1);
   cout << "Epsilon DW =\t" << epsilon << endl;

//for
   x=1;
   for(;1;)  {
        x/=2;
        if(x+1==1) break;
   }
   epsilon=x*2;
   cout << "Epsilon for =\t" << epsilon << endl;
//
    return 0;
}
```

```
vinces-MacBook-Pro:Debug vm2$ ./machineprecision
Epsilon while = 2.22045e-16
Epsilon DW =    2.22045e-16
Epsilon for =   2.22045e-16
vinces-MacBook-Pro:Debug vm2$
```

TRAPEZOID RULE

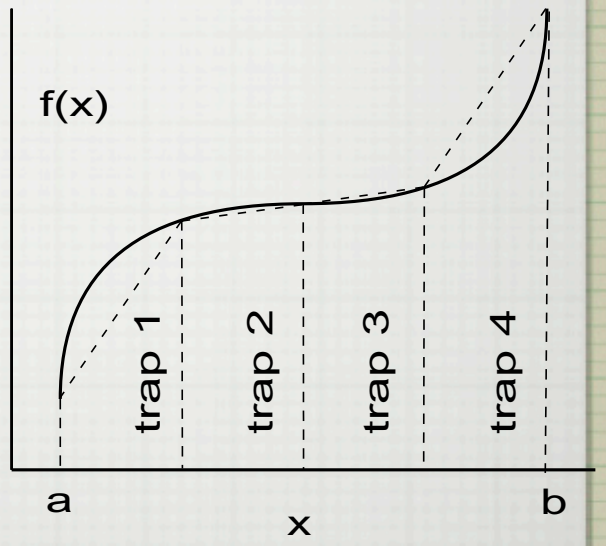


TRAPEZOID RULE



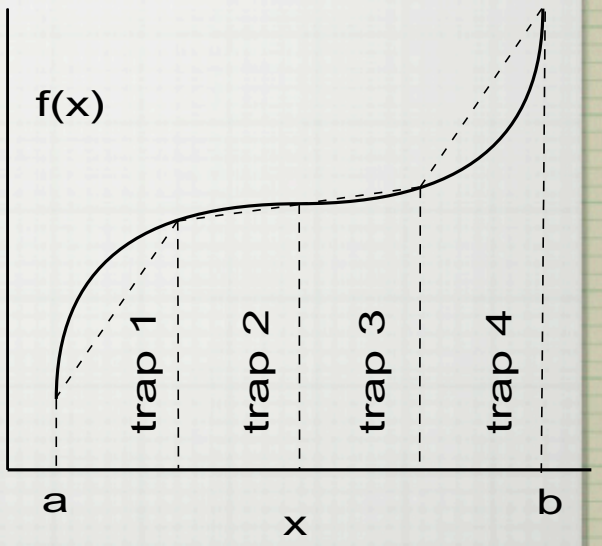
TRAPEZOID RULE

- **Values of f(x) at evenly spaced values of x (N values x_i)**



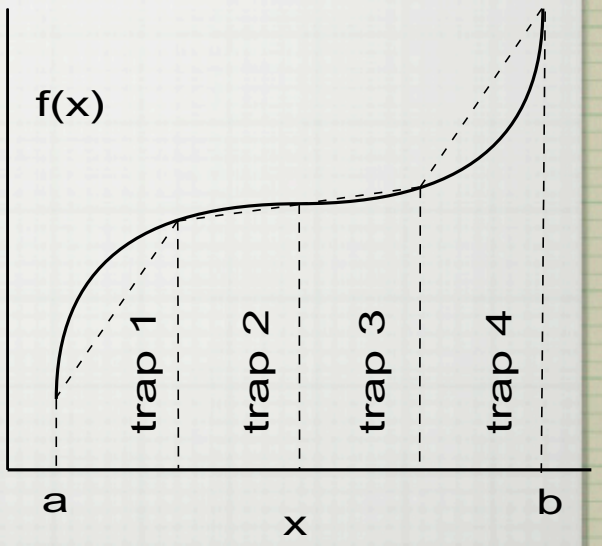
TRAPEZOID RULE

- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)



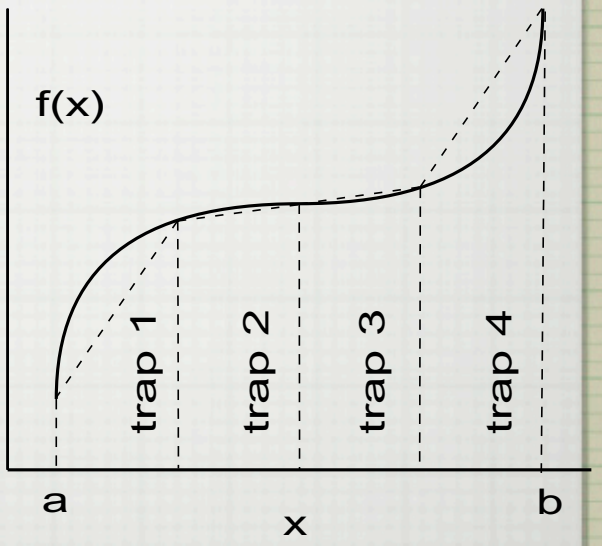
TRAPEZOID RULE

- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)
- N-1 intervals of length h



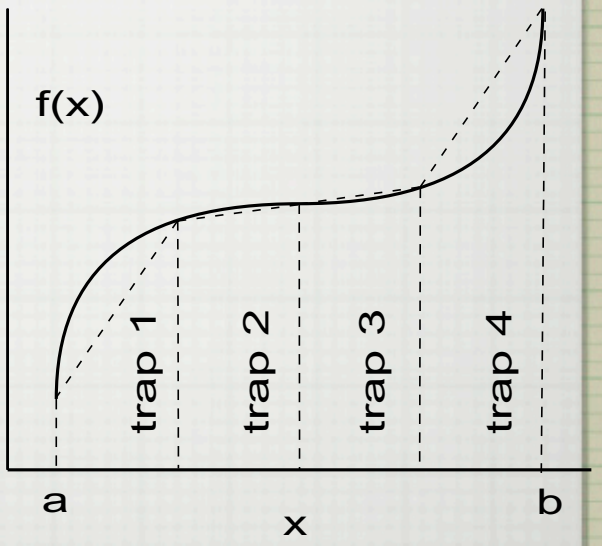
TRAPEZOID RULE

- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)
- N-1 intervals of length h



TRAPEZOID RULE

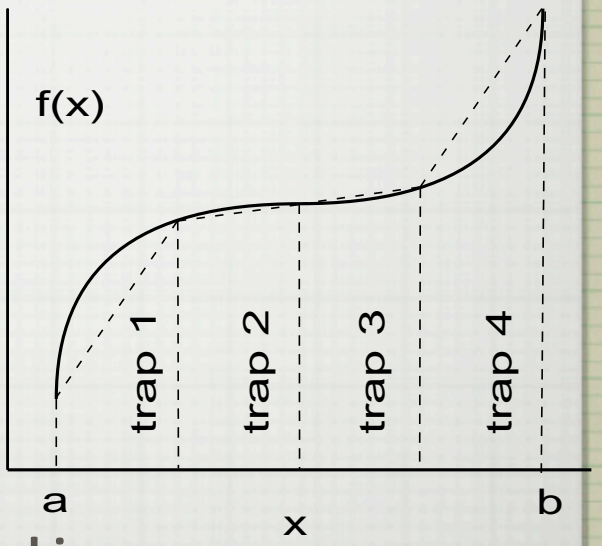
- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)
- N-1 intervals of length h



TRAPEZOID RULE

- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)
- N-1 intervals of length h

$$h = \frac{b-a}{N-1} \qquad x_i = a + (i-1)h \quad i = 1, N$$

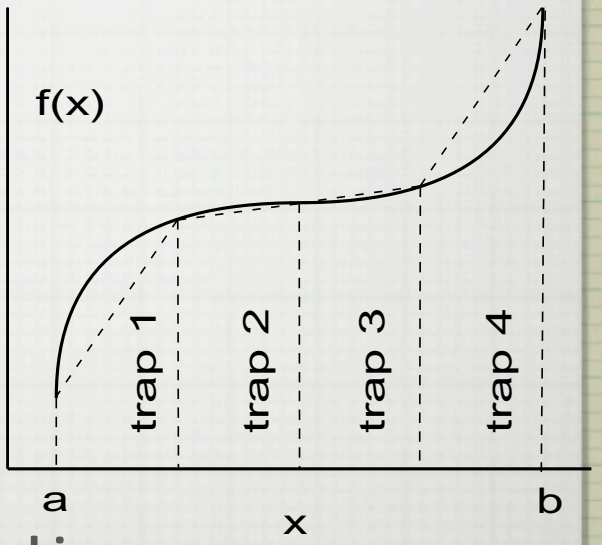


- This approximates f(x) by a straight line in that interval i, and uses the average height $(f_i + f_{i+1})/2$ as the value for f

TRAPEZOID RULE

- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)
- N-1 intervals of length h

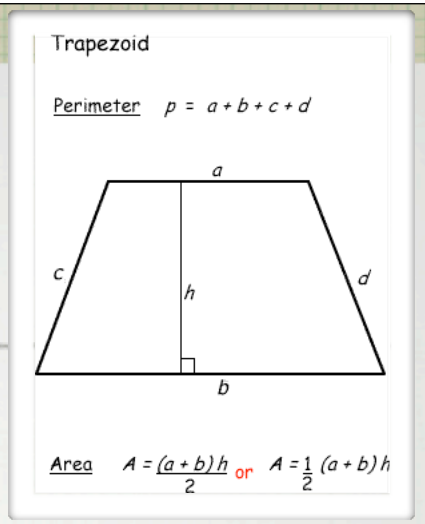
$$h = \frac{b-a}{N-1} \qquad x_i = a + (i-1)h \quad i = 1, N$$



- This approximates f(x) by a straight line in that interval i, and uses the average height ($f_i + f_{i+1}$)/2 as the value for

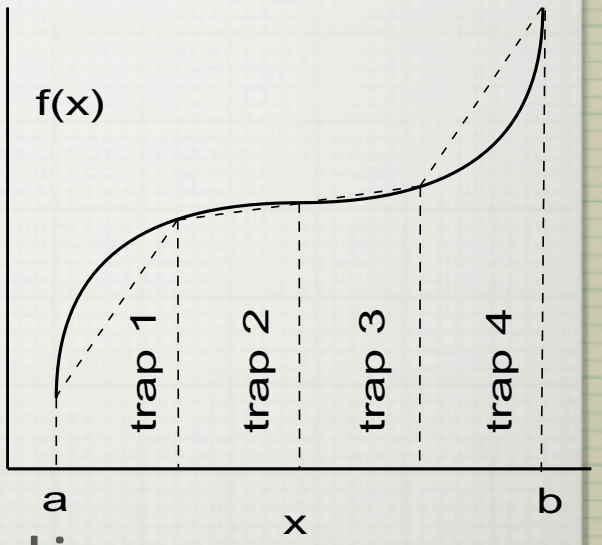
$$\int_{x_i}^{x_i+h} f(x)dx \simeq \frac{h(f_i + f_{i+1})}{2} = \frac{1}{2}hf_i + \frac{1}{2}hf_{i+1}$$

TRAPEZOID RULE



- Values of f(x) at evenly spaced values of x (N values x_i)
- The N values include the endpoints (N must be odd!)
- N-1 intervals of length h

$$h = \frac{b-a}{N-1} \qquad x_i = a + (i-1)h \quad i = 1, N$$



- This approximates f(x) by a straight line in that interval i, and uses the average height $(f_i + f_{i+1})/2$ as the value for I

$$\int_{x_i}^{x_i+h} f(x)dx \simeq \frac{h(f_i + f_{i+1})}{2} = \frac{1}{2}hf_i + \frac{1}{2}hf_{i+1}$$

NUMERICAL INTEGRATION: TRAPEZOID

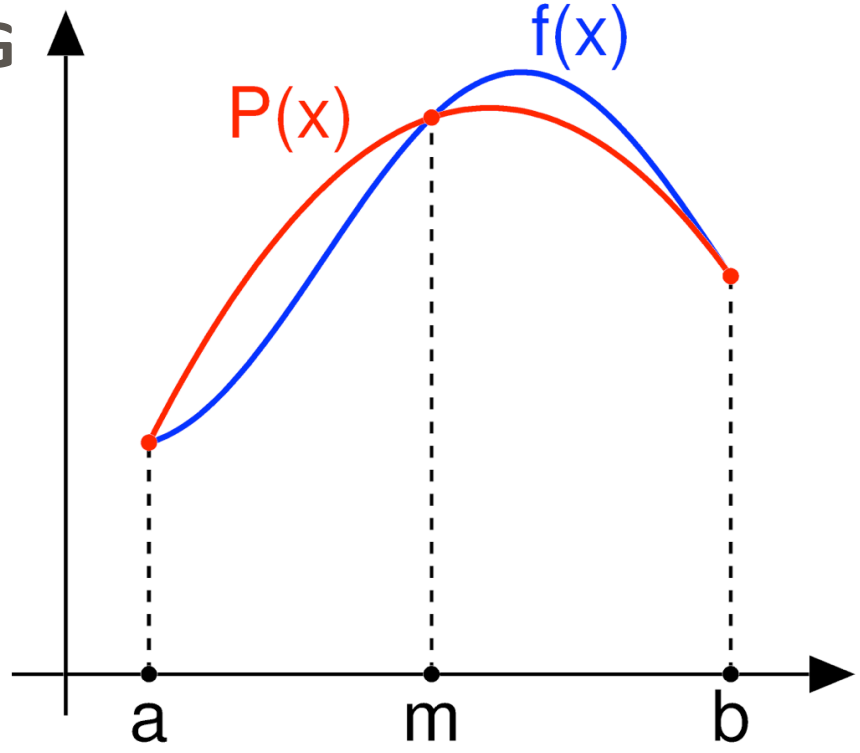
```
#include <iostream>
using namespace std;

int main (){
    int n = 21;
    int i;
    double f[n]; //where we store the function
    double w[n]; //where we store the weight
    double x[n]; //where we store the abscissas
    double integral; //where we store the integral

//define the weights
    for(i=0;i<n;i++){
         w[i]=1.0;
    }
    w[0]=0.5;
    w[n-1]=0.5;

//define function
    for(i=0;i<n;i++){
         x[i]=(double) i/ (double) (n-1); //varies between 0 and 1
         f[i]=x[i]*x[i]; //here we could define f1, f2, f3, ... etc
    }
// compute integral
    integral=0.0;
    for(i=0;i<n;i++){
         integral+=w[i]*f[i];
    }
// finalize integration
    integral=integral/(n-1);
    cout << "Integral=" << integral << endl;
    return 0;
}
```

SUMMARY OF SIMPSON'S RULE

SIMPSON'S RULE CAN BE DERIVED BY APPROXIMATING THE INTEGRAND F (X) (IN BLUE) BY THE QUADRATIC INTERPOLANT P (X) (IN RED).

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

$$\text{ERROR} \approx \frac{(b-a)^5}{180n^4}\left|f^{(4)}(\xi)\right|$$



NUMERICAL INTEGRATION: SIMPSON

```
#include <iostream>
#include <math.h>
using namespace std;

int main (){
    int n = 3;
    int i;
    double f[n]; //where we store the function
    double w[n]; //where we store the weight
    double x[n]; //where we store the abscissas
    double integral; //where we store the integral

//define the weights
//we need to be careful: each end of the 3-point interval must be counted twice
    for(i=0;i<n;i=i+2){
         w[i]=2.0;
    }
    for(i=1;i<n;i=i+2){
         w[i]=4.0;
    }
    w[0]=1.0;
    w[n-1]=1.0;

//define function
    for(i=0;i<n;i++){
         x[i]=(double) i/ (double) (n-1); //varies between 0 and 1
         f[i]=x[i]*x[i]*x[i]; //here we could define f1, f2, f3, ... etc
    }
// compute integral
    integral=0.0;
    for(i=0;i<n;i++){

         integral+=w[i]*f[i];
    }
// finalize integration
    integral=integral/(3.0*(n-1)); //this pre-term 3*(n-1) comes from Simpson's rule:
                                   //3 points for each interval
                                   // n-1 intervals for n points
    cout << "Integral=" << integral << endl;

     return 0;
}
```

N-POINT GAUSS QUADRATURE RULE

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

N-POINT GAUSS QUADRATURE RULE

- **General formulation:**

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

N-POINT GAUSS QUADRATURE RULE

- **General formulation:**

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

N-POINT GAUSS QUADRATURE RULE

- **General formulation:**

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

N-POINT GAUSS QUADRATURE RULE

- **General formulation:**

- **Problem:**

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

N-POINT GAUSS QUADRATURE RULE

- **General formulation:**

- **Problem:**

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

N-POINT GAUSS QUADRATURE RULE

- **General formulation:**

$$\int_{-1}^{1} g(x)dx \approx \sum_{i=1}^{n} c_i g(x_i)$$

- **Problem:**

So if the table is given for $\int_{-1}^{1} g(x)dx$ integrals, how does one solve $\int_{a}^{b} f(x)dx$?

- **Solution:**

$$\int_{a}^{b} f(x)dx = \int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right)\frac{b-a}{2}dx$$

Points	Weighting Factors	Function Arguments
2	$c_1 = 1.000000000$	$x_1 = -0.577350269$
	$c_2 = 1.000000000$	$x_2 = 0.577350269$
3	$c_1 = 0.555555556$	$x_1 = -0.774596669$
	$c_2 = 0.888888889$	$x_2 = 0.000000000$
	$c_3 = 0.555555556$	$x_3 = 0.774596669$
4	$c_1 = 0.347854845$	$x_1 = -0.861136312$
	$c_2 = 0.652145155$	$x_2 = -0.339981044$
	$c_3 = 0.652145155$	$x_3 = 0.339981044$
	$c_4 = 0.347854845$	$x_4 = 0.861136312$
5	$c_1 = 0.236926885$	$x_1 = -0.906179846$
	$c_2 = 0.478628670$	$x_2 = -0.538469310$
	$c_3 = 0.568888889$	$x_3 = 0.000000000$
	$c_4 = 0.478628670$	$x_4 = 0.538469310$
	$c_5 = 0.236926885$	$x_5 = 0.906179846$
6	$c_1 = 0.171324492$	$x_1 = -0.932469514$
	$c_2 = 0.360761573$	$x_2 = -0.661209386$
	$c_3 = 0.467913935$	$x_3 = -0.238619186$
	$c_4 = 0.467913935$	$x_4 = 0.238619186$

NUMERICAL INTEGRATION: GAUSSIAN

```
#include <iostream>
#include <math.h>
using namespace std;

int main (){
    int n = 3;
    int i;
    double f[n]; //where we store the function
    double w[n]; //where we store the weight
    double x[n]; //where we store the abscissas
    double integral; //where we store the integral

//define the weights and abscissas
    x[0]=-0.774596669;
    x[1]=0.0;
    x[2]=0.774596669;

    w[0]=5.0/9.0;
    w[1]=8.0/9.0;
    w[2]=5.0/9.0;
//define function
    for(i=0;i<n;i++){
         f[i]=x[i]*x[i]*x[i]*x[i];
     }
// compute integral
    integral=0.0;
    for(i=0;i<n;i++){
         integral+=w[i]*f[i];
    }
    cout << "Integral=" << integral << endl;

     return 0;
}
```