Short introduction to C++
- Structure of a program
- Variables, Data Types, and Constants
- Operators
- Basic Input/Output
- Control Structures
- Functions
- Arrays
- Input/Output with files
- Pointers
- Classes

Reference books

Good practice
Have a good reference book for the version of C++ you are using.
Refer to this book frequently to be sure you are aware of the rich collection of C++ features and you are using these features correctly.

Programming tips
Some books* have very practical advice on
# Good programming practices
# Common programming errors
# Performance tips
# Software engineering observations
# Testing and debugging tips

* C++ how to program, Deitel & Deitel have hundreds of valuable tips.

Part 1
Structure of a program
// Simple program
#include <iostream>
using namespace std;

int main()
{
    int x, y;
    x = 2;
    y = x + 4;
    cout << "x = \"" << x << \"x + 4 = \"" << y << \"end1;"
    return 0;
}

Free-format language

C++ is a free-format language like many other languages.

The compiler ignores ALL spaces, tabs, and new-line characters (also called “white spaces”)

The compiler recognizes “white spaces” only inside a string.

Using white spaces allows to better visualize a program structure (e.g. extra indentation inside if statements, for loops, etc.)

Common structure of a program

1. Comments
2. Header files
3. Declare variables
4. Declare constants
5. Read initial data
6. Open files
7. CALCULATIONS (include calling other functions)
8. Write results
9. Closing
10. Stop

Steps 5 – 9 may call other modules

Variables, Data Types and Constants

- Identifiers (names of variables)
- Fundamental data types
- Declaration of variables
- Global and local variables
- Initialization of variables
- Constants

Variables

Variable is a location in the computer’s memory where a value can be stored for use by a program.
**Identifiers – Names of variables**

A variable name is any valid identifier. An identifier is a series of characters consisting of letters, digits, and underscore (_) that does not begin with a digit.

C++ is case sensitive – uppercase and lowercase letters are different.

Examples:

- abc
- velocity_i
- Force_12

**Identifiers: reserved key words**

These keywords must not be used as identifiers!

C and C++ keywords

- auto
- break
- case
- char
- const
- continue
- default
- do
- double
- else
- enum
- extern
- float
- for
- goto
- if
- int
- long
- register
- return
- short
- signed
- sizeof
- static
- struct
- switch
- typedef
- union
- unsigned
- volatile
- while
- void

**Identifiers: reserved key words II**

C++ only keywords

- asm
- bool
- catch
- class
- const_cast
- delete
- dynamic_cast
- explicit
- false
- friend
- inline
- mutable
- namespace
- new
- operator
- private
- protected
- public
- reinterpret_cast
- static_cast
- template
- this
- throw
- true
- try
- typeid
- typename
- using
- virtual
- wchar_t

**Variables: Data Types**

Each variable has a name, a type, a size and a value.

**Fundamental data types in C++**

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>Character or small integer</td>
<td>1</td>
</tr>
<tr>
<td>short int</td>
<td>Short Integer</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>4</td>
</tr>
<tr>
<td>long int</td>
<td>Long integer</td>
<td>4*</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean</td>
<td>1</td>
</tr>
<tr>
<td>float</td>
<td>Floating point number</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>Double precision floating point</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>Long double precision</td>
<td>8*</td>
</tr>
<tr>
<td>wchar_t</td>
<td>Wide character</td>
<td>2</td>
</tr>
</tbody>
</table>

* depends on a system

**Range of data types in C++**

<table>
<thead>
<tr>
<th>name</th>
<th>range</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>short int</td>
<td>signed: -32768 to 32767</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>unsigned: 0 to 65535</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>-2147483648 to 2147483647</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>unsigned: 0 to 4294967295</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>true or false</td>
<td>1</td>
</tr>
<tr>
<td>float</td>
<td>3.4e +/- 38 (7 digits)</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>1.7e +/- 308 (15 digits)</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>1.7e +/- 308 (15 digits)</td>
<td>8*</td>
</tr>
</tbody>
</table>
Declaration of variables

All variables must be declared with a name and a data type before they can be used by a program.

```
#include <iostream>
using namespace std;
int main()
{
    double a, speed, force_12;
    int i, n;
    ... some operators ...
    return 0;
}
```

Global and local variables

A global variable is a variable declared in the main body of the source code, outside all functions. Global variables can be referred from anywhere in the code, even inside functions.

A local variable is one declared within the body of a function or a block. The scope of local variables is limited to the block enclosed in braces {} where they are declared.

```
#include <iostream>
using namespace std;
void f12(void);
int nglobal = 1;
main()
{
    cout << "main 1: nglobal = " << nglobal <<endl;
    nglobal = 2;
    cout << "main 2: nglobal = " << nglobal <<endl;
    f12();
    cout << "main 3: nglobal = " << nglobal <<endl;
}
void f12()
{
    cout << "f12 : nglobal = " << nglobal <<endl;
    nglobal = 3;
}
```

Initialization of variables

When declaring a regular local variable, its value is by default undetermined.

Initialization 1:
```
type identifier = initial_value;
```

```
float sum = 0.0;
```

Initialization 2:
```
type identifier (initial_value);
```

```
float sum (0.0);
```

Constants

Declared constants
```
const type identifier = initial_value ;
```

Constant variable can not be modified thereafter.

```
const double pi = 3.1415926;
```

Define constants
```
#define identifier value
```

```
#define PI 3.14159265
```

Example

```
#include <iostream>
#include <iofstream>
using namespace std;
#define PI 3.14159265
const float Ry = 13.6058;
int main()
{
    float a, speed, force_12;
    int i, n;
    float angle = 45.0;
    ... some operators ...
    return 0;
}
```
Part 3
Operators

Operators

- Assignment (=)
- Arithmetic operators (+, - , *, /, % )
- Compound assignation (+=, -=, *=, /=, %=)
- Increment and decrement (++, --)
- Relational and equality operators ( ==, !=, >, <, >=, <=)
- Logical operators (!, &&, ||)
- Conditional operator (?)
- Comma operator (,)
- Precedence of operators

Assignment operator (=)
The assignment operator assigns a value to a variable.

```cpp
// operator (=)
#include <iostream>
using namespace std;
int main ()
{
    int a, b;
    a = 12;
    b = a;
    cout << " a = " << a << " b = " << b <<endl;
    return 0;
}
```

Arithmetic operators
There are five arithmetic operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>C++ example</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>+</td>
<td>f + 7</td>
</tr>
<tr>
<td>subtraction</td>
<td>-</td>
<td>p - c</td>
</tr>
<tr>
<td>multiplication</td>
<td>*</td>
<td>b * k</td>
</tr>
<tr>
<td>division</td>
<td>/</td>
<td>x / y</td>
</tr>
<tr>
<td>modulus</td>
<td>%</td>
<td>r % s</td>
</tr>
</tbody>
</table>

Arithmetic assignment operators
There are five arithmetic assignment operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>C++ explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>a += 7</td>
</tr>
<tr>
<td>-=</td>
<td>b -= 4</td>
</tr>
<tr>
<td>*=</td>
<td>c *= 5</td>
</tr>
<tr>
<td>/=</td>
<td>d /= 3</td>
</tr>
<tr>
<td>%=</td>
<td>e %= 9</td>
</tr>
</tbody>
</table>
### The increment/decrement operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>C++ syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>++a</td>
<td>pre increment</td>
</tr>
<tr>
<td>++</td>
<td>a++</td>
<td>post increment</td>
</tr>
<tr>
<td>--</td>
<td>--a</td>
<td>pre decrement</td>
</tr>
<tr>
<td>--</td>
<td>a--</td>
<td>post decrement</td>
</tr>
</tbody>
</table>

### Equality and relational operators

**Equality operators in decision making**

<table>
<thead>
<tr>
<th>Operator</th>
<th>C++ syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>x == y</td>
<td>x is equal to y</td>
</tr>
<tr>
<td>!=</td>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
</tbody>
</table>

**Relational operators in decision making**

<table>
<thead>
<tr>
<th>Operator</th>
<th>C++ syntax</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>&gt;=</td>
<td>x &gt;= y</td>
<td>x is greater or equal to y</td>
</tr>
<tr>
<td>&lt;=</td>
<td>x &lt;= y</td>
<td>x is less than or equal to y</td>
</tr>
</tbody>
</table>

### Logical operators

C++ provides **logical operators** that are used to form complex conditions by combining simple conditions.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbols</th>
<th>C++ example</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>&amp;&amp;</td>
<td>if (i==1 &amp;&amp; j&gt;=10)</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Condition operator ?

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format

```
condition ? result1 : result2
```

```cpp
// conditional operator
#include <iostream>
using namespace std;

int main ()
{
    int a,b,c;
    a=2;
    b=7;
    c = (a>b) ? a : b;
    cout << " c = " << c;
    return 0;
}
```

Part 4

Basic Input/Output
Input/Output

The C++ libraries provide an extensive set of input/output capabilities.

C++ I/O occurs in stream of bytes.

Iostream Library header files

<iostream.h> contains cin, cout, cerr, clog.
<iomanip.h> information for formatting
<ifstream.h> for file processing

Basic Input/Output

**cin** is an object of the istream class and is connected to the standard input device (normally the keyboard)

**cout** is an object of the ostream class and is connected to the standard output device (normally the screen)

Example (output)

```cpp
// output
#include <iostream.h>
int main ()
{
    int a;
    a=2;
    cout << " a = " << a << endl;
    return 0;
}
```

```
a = 2
```

Example (input/output)

```cpp
// Input and output
#include <iostream.h>
int main ()
{
    int a, b;
    cout << " enter two integers:";
    cin >> a >> b;
    cout << " a = " << a << " b = " << b << endl;
    return 0;
}
```

```
enter two integers:2 4
a = 2 b = 4
```

Elements of formatting

**setw** set the field width (positions for input/output)

**setprecision** control the precision of float-point numbers

**setiosflags**(ios::fixed | ios::showpoint) sets fixed point output with a decimal point

```cpp
cout << setw(5)< a << setw(10)< setprecision(4)< setiosflags(ios::fixed | ios::showpoint)< t <<endl;
```

```
for n = 2 and t = 4.0
```

Some format state flags

**ios::showpoint** Specify that floating-point numbers should be output with a decimal

**ios::fixed** Specify output of a floating-point value in fixed-point notation with a specific number of digits to the right of the decimal point.

**ios::scientific** Specify output of a floating-point value in scientific notation.

**ios::left** Left justify output in a field.

**ios::right** Right justify output in a field.
Example

```cpp
cout.setf(ios::fixed | ios::showpoint);
cout.width(10);
cout.precision(5);
cout << "radius = " << radius << endl;
cout << "diameter = " << diameter << endl;
cout << "circumf. = " << circumf << endl;
cout << "area = " << area << endl;
```

radius = 3.00000
diameter = 6.00000
circumf. = 18.84956
area = 28.27433

Part 5
Control Structures

Control Structures

Normally, statements in a program are executed one after another in the order in which they are written. This is called sequential execution.

The transfer of control statements enable the programmer to specify that the next statement to be executed may be other than the next one in the sequence.

Three types of selection structures:

- if single-selection structure
- if/else double-selection structure
- switch multiple-selection structure

if - single-selection structure

The if selection structure performs an indicated action only when the condition is true; otherwise the condition is skipped.

```cpp
if (grade >=60) {
    cout << "passed";
    if (grade >=60) {
        n=n+1;
        cout << "passed";
    }
```
if/else - double-selection structure

The if/else selection structure allows the programmer to specify that a different action is to be performed when the condition is true than when the condition is false.

```
if (grade >= 60)
  cout << "passed";
else
  cout << "failed";
```

switch - multiple-selection structure

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

Three types of repetition structures:

- while
- do/while
- for

The while repetition structure

A repetition structure allows the programmer to specify an action is to be repeated while some condition remains true.

```
int n = 2;
while (n <= 100)
  {n = 2 * n;
   cout << n;
  }
```

The do/while repetition structure

The loop-continuation condition is not executed until after the action is performed at least once.

```
do/while structure

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

```
int i = 0;
do {
  cout << i;
  i = i + 10;
} while (i < 100);
```

The for repetition structure

The for repetition structure handles all the details of counter-controlled repetition.

```
for structure

for (i=0; i <= 5; i=i+1) {
  ~actions~
}
```
The break and continue statements

The break and continue statements alter the flow of control.

The break statement, when executed in a while, for, do/while, or switch structure, causes an immediate exit from that structure.

The continue statement, when executed in a while, for, or do/while structure, skips the remaining statements in the body of the structure, and proceeds with the next iteration.

```c++
// using the break statement
#include <iostream.h>
int main ()
{
  int n;
  for (n = 1; n <= 10; n = n+1)
  {
    if (n == 5)
      break;
    cout << n << " ";
  }
  cout << "Broke out of loop at n of " << n;
  return 0;
}
```

```
1 2 3 4 
Broke out of loop at n of 5
```

// Using the continue statement
```c++
#include <iostream.h>
int main()
{
  for ( int x=1; x<=10; x++)
  {
    if (x == 5)
      continue;
    cout << x << " ";
  }
  cout << "Used continue to skip printing 5" << endl;
  return 0;
}
```

```
1 2 3 4 6 7 8 9 10 
Used continue to skip printing 5
```

Good practice:

The while structure is sufficient to provide any form of repetition.

Functions

The best way to develop and maintain a large program is to construct it from smaller parts (modules).

Modules in C++ are called functions and classes.

C++ standard library has many useful functions.

Functions written by a programmer are programmer-defined-functions.
Math Library Functions

Math library functions allows to perform most common mathematical calculations

Some math library functions:
- \( \cos(x) \)
- \( \sin(x) \)
- \( \tan(x) \)
- \( \sqrt{x} \)
- \( \exp(x) \)
- \( \log(x) \)
- \( \log10(x) \)
- \( \text{pow}(x,y) \)
- \( \text{fabs}(x) \)
- \( \text{floor}(x) \)
- \( \text{fmod}(x,y) \)
- \( \text{ceil}(x) \)

Header files

Each standard library has a corresponding header file containing the function prototypes for all functions in that library and definitions of various types and constants

Examples

old styles and new styles

- \(<\text{math.h}> \quad <\text{cmath}> \quad \text{math library}\)
- \(<\text{iostream.h}> \quad <\text{iostream}> \quad \text{input and output}\)
- \(<\text{fstream.h}> \quad <\text{fstream}> \quad \text{read and write (disk)}\)
- \(<\text{stdlib.h}> \quad <\text{cstdlib}> \quad \text{utility functions}\)

... and many more

Examples

old style

```c
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <math.h>
```

new style (note – add a line)

```c
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
```

Functions prototypes

A function-prototype tells the compiler the name of the function, the type of data returned by the function, the number of parameters, the type of parameters, and the order of parameters.

Function prototype:

```
value-type function-name (par-type1, par-type2, …)
```

The compiler uses function prototypes to validate function calls.

Functions definitions

Function definition:

```
return-value-type function-name(parameter-list)
{
 declarations and statements (function body)
}
```

A type must be listed explicitly for each parameter in the parameter-list of a function

All variables declared in function definitions are local variables – they are known only in the function.

```c
//example: a programmer-defined function
#include <iostream.h>
int square( int ); // function prototype
int main()
{
 for ( int x = 1; x <= 10; x++ )
   cout << square( x ) << " ";
 cout << endl;
 return 0;
}
```

```c
// Function definition
int square( int y )
{
 int result;
 result = y * y;
 return result;
}
```

```
1  4  9  16  25  36  49  64  81  100
```

Functions definitions

If a function does not receive any values, the parameter-list is void or left empty.

If a function does not return any value, then the return-value-type of that function is void both in the function prototype and function definition.

```
#include <iostream.h>
void out2(void);   // function prototype
int main()
{
    out2();
    return 0;
}
// Function definition
void out2(void)
{
    cout << "output from function out2";
    return;
}
```

output from function out2

References and Reference Parameters

There are two ways to invoke functions:

*call-by-value* – a copy of the argument's value is made and passed to the called function. Changes to the copy do not affect the original variable's value in the caller. (This is the common way)

*call-by-reference* – the caller gives the called function the ability to directly access the caller's data, and to modify that data if the called function so chooses.

```
#include <iostream.h>
void f12(int&, int&);
int main()
{
    int a, b;
    a = 12;
    b = a;
    cout << "a = " << a << "  b = " << b << endl;
    f12(a, b);
    cout << "a = " << a << "  b = " << b << endl;
    return 0;
}
void f12(int& out1, int& out2)
{
    out1 = out1*2.0;
    out2 = out1 +3;
}
```

call-by-reference

A reference parameter is an alias for the corresponding argument.

To indicate that place & after the parameter's type in the function prototype, and the function definition.

```
void function2(int a=2);  // default argument
```
Part 7

Arrays

An array is a consecutive group of memory locations that all have the same name and the same type.

To refer to a particular location or element in the array, we specify the name of the array and the position number of the particular element in the array.

The first element in the every array is the 0th element.

Arrays in C/C++

Most of us were not taught by our mothers to count on our fingers starting with the thumb as zero!

Accordingly, you will probably make fewer n - 1 errors if you do not use zero subscripts when dealing with matrices.

F.S. Acton "Real Computing made real"

Arrays in C/C++

Most of us were not taught by our mothers to count on our fingers starting with the thumb as zero!

Accordingly, you will probably make fewer n - 1 errors if you do not use zero subscripts when dealing with matrices.

F.S. Acton "Real Computing made real"

Declaring Arrays

Arrays occupy space in memory. The programmer specifies the type of elements and the number of elements required, so that the compiler may reserve the appropriate amount of memory.

Example: reserve 12 elements for integer array c

```
int c[12];
```

Example: declaration and initialization of an array n

```
int n[6]={2, 18, 33, 5, 21, 39};
```

// Initialize array a and fill with numbers
#include <iostream.h>
#include <iomanip.h>
int main()
{
    const int arraySize = 5;
    int i, a[ arraySize ];
    for ( i = 0; i < arraySize; i = i + 1 )
        a[ i ] = 2 * i;
    cout <<"Element"<<setw(12)<<"Value"<< endl;
    for ( i = 0; i < arraySize; i = i + 1 )
        cout <<setw(7)<<i<<setw(12)<<a[ i ]<<endl;
    return 0;
}

Multidimensional Arrays

Example: A 2 dimensional table 3 (rows) by 5 (columns) (15 elements)

```
int toys[3][5];
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>4</td>
<td>21</td>
<td>0</td>
</tr>
</tbody>
</table>

toys [2][3] = 21;
Passing Arrays to Functions

To pass an array argument to a function, specify the name of the array without any brackets.

Example for array `time` and function `speed`.

```cpp
float array time[24];
... speed(time, 24);
```

C++ passes arrays to functions using simulated call-by-reference – the called function can modify the element values in the caller’s original arrays.

Static and Automatic Arrays

Arrays that are declared `static` are initialized when the program is loaded. If a `static` array is not explicitly initialized, that array is initialized to zero by the compiler.

In functions: static arrays contain the values stored during the previous function call. For automatic arrays it does not happen.

```cpp
static int array_s[10];
int array_a[10];
```

File processing (open and write)

To perform file processing in C++, the header files `<iostream.h>` and `<fstream.h>` must be included.

```cpp
#include <iostream.h>
#include <fstream.h>
ofstream outfile ("file1.dat", ios::out);
... outfile << a; outfile << endl;
```

Part 8

Input/Output with files
File processing (more)

Example 2 (also works)
Open a file with a name “file2.dat” and write to it

```cpp
#include <iostream.h>
#include <fstream.h>
ofstream outfile;
outfile.open("file2.dat");
outfile << a << endl;
```

File processing (open and read)

Open a file with a name “input.dat” and read from it

```cpp
#include <iostream.h>
#include <fstream.h>
ifstream inputfile ("input.dat", ios::in);
inputfile >> a;
```

To close a file

```cpp
inputfile.close();
```

File open modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ios::app</td>
<td>Write all output to the end</td>
</tr>
<tr>
<td>ios::in</td>
<td>Open a file for input</td>
</tr>
<tr>
<td>ios::out</td>
<td>Open a file for output</td>
</tr>
<tr>
<td>ios::nocreate</td>
<td>If the file does not exist, the open operation fails</td>
</tr>
<tr>
<td>ios::noreplace</td>
<td>If the file exists, the open operation fails</td>
</tr>
</tbody>
</table>

Part 9

Pointers

Pointers are one of the most powerful features of the C++ programming language.
Pointers are among the most difficult capabilities to master.
Pointers enable to simulate call by reference, and to create and manipulate dynamic data structures.

Declarations

Pointer variables contain memory address as their values.

```cpp
int *iPointer, i;
float *xPointer, x;
double *zpntr;
```
**Pointer operations**

Important: & is **address operator** that returns the address of its operand

```cpp
int y = 5;
int *yptr;
```

the statement

```cpp
yptr = &y;
```

assigns the address of the variable y to pointer yptr

Now the statement

```cpp
cout << *yptr << endl;
```

print the value of y, namely 5.

And the statement

```cpp
*yptr = 9;
```

would assign 9 to y.

---

**Function pointers**

A pointer to a function contains the address of the function in memory.

A function name is the starting address in memory of the code that performs the function’s task

Pointers to functions can be processed to functions, returned to functions, stored in arrays, and assigned to other function pointers.

---

**Examples**

```cpp
#include <iostream.h>

int main()
{
    int number = 5;
    cout << "The side is " << number;
    cubeByReference( &number );
    cout << "The volume is " << number << endl;
    return 0;
}

cubeByReference( int *nPtr )
{
    *nPtr = *nPtr * *nPtr * *nPtr;
}
```

```
#include <iostream.h>

float av( float a, float b, float (*f)(float))
{
    return (f(b)+f(a))/2.0;
}

float x2 (float x)
{
    return x*x;
}
```

---

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>

using namespace std;

double f(double);        //function prototype

int main()
{
    const double pi=3.1415926;
    double a, b, step, x, y;
    int i, n;
    ofstream out2disk;    //output to out2disk
    // Example 1: calculate values of a function and write to a file
    see the next slide ...
    ```
a = 0.0; // left endpoint
b = 2.0*pi; // right endpoint
n = 12; // number of points
step = (b-a)/(n-1);
out2disk.open("table01.dat");
out2disk <<"     x" <<"       f(x)"<< endl;
i=1;
while (i <= n)
{x = a + step*(i-1);
y = f(x);
out2disk << setw(12) << setprecision(5)
      << setiosflags(ios::fixed|ios::showpoint)
      << x << setw(12) << setprecision(5)
      << setiosflags(ios::fixed|ios::showpoint)
      << y <<endl;
  i = i+1;
}
return 0;

// Function f(x)
double f(double x)
{
double y;
y = sin(x);
return y;
}

<table>
<thead>
<tr>
<th>x</th>
<th>f(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>0.57120</td>
<td>0.54064</td>
</tr>
<tr>
<td>1.14240</td>
<td>0.90963</td>
</tr>
<tr>
<td>1.71360</td>
<td>0.98982</td>
</tr>
<tr>
<td>2.28479</td>
<td>0.75575</td>
</tr>
<tr>
<td>2.85599</td>
<td>0.28173</td>
</tr>
<tr>
<td>3.42719</td>
<td>-0.28173</td>
</tr>
<tr>
<td>3.99839</td>
<td>-0.75575</td>
</tr>
<tr>
<td>4.56959</td>
<td>-0.98982</td>
</tr>
<tr>
<td>5.14079</td>
<td>-0.90963</td>
</tr>
<tr>
<td>5.71199</td>
<td>-0.54064</td>
</tr>
<tr>
<td>6.28319</td>
<td>-0.00000</td>
</tr>
</tbody>
</table>

see the next slide …