OLD DOMINION UNIVERSITY

C/C++ Short review

A. Godunov

1. Structure of a program
2. Variables, Data Types, and Constants
3. Operators
4. Basic Input/Output
5. Control structures
6. Functions
7. Arrays
8. Input/Output with files
9. Pointers

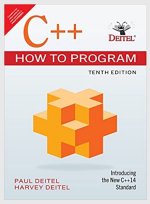
1

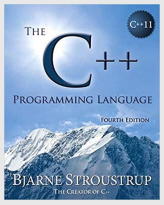
Basics of C/C++ for computational physics

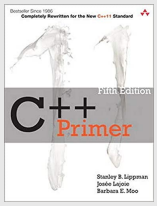
- Structure of a program
- Variables, Data Types, and Constants
- Operators
- Basic Input/Output
- Control Structures
- Functions
- Arrays
- Input/Output with files
- Pointers
- Classes

2

Reference books: so many!







Have a good reference book for the version of C/C++ you are using.

Refer to this book frequently to be sure you are aware of the rich collection of C/C++ features and you are using these features correctly.

3

Books with 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.

4

Part 1: Structure of a program

5

A simple program

```
// Simple program
#include <iostream>
using namespace std;
int main()
{
    int x, y;
    x = 2;
    y = x + 4;
    cout <<" x = "<<x<<"  x + 4 = "<<y << endl;
    return 0;
}

Output:
 x = 2  x + 4 = 6
```

More complex structure involves programmer-defined functions, control statements, classes, communication with files, …

6

C/C++ is a free format language

- 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.) .

7

7

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

8

8

Part 2: Variables, Data Types, and Constants

9

Variables, Data Types and Constants

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

10

10

Variables and Identifiers

Variable is a location in the computer’s memory where a value can be stored for use by a program.

A variable name is any valid identifier.

An identifier is a series of characters consisting of letters, digits, and uderscore (_) that does not begin with a digit.

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

Examples: `abc, Velocity_i, Force_12`

11

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	void
volatile	while			

12

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				

13

Variable Data Types

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

Fundamental data types in C++

Name	Description	Bytes
char	Character or small integer	1
short int	Short Integer	2
int	Integer	4
long int	Long integer	4*
Long long int	Long integer	8
bool	Boolean	1
float	Floating point number	4
double	Double precision floating point	8
long double	Long double precision	8*
wchar_t	Wide character	2

* Depends on a system

14

14

Range of data types in C++

name	range	bytes
short int	signed: -32768 to 32767 unsigned: 0 to 65535	2
int	-2,147,483,648 to 2,147,483,647 unsigned: 0 to 4,294,967,295	4
bool	true or false	1
float	3.4e +/- 38 (7 digits)	4
double	1.7e +/- 308 (15 digits)	8
long double	1.7e +/- 308 (15 digits)	8*

* Depends on a system

15

C++ and complex numbers

C++, unlike Fortran, does not have complex numbers as a part of the language. However, there are libraries

```
#include <complex>
// Program illustrating the use of real() and
// imag() function
#include <iostream>
#include <complex>
using namespace std;
// main part
int main()
{
// defines the complex number: (10 + 2i)
std::complex<double> mycomplex(10.0, 12.0);
// prints the real part using the real function
cout << "Real:      " << real(mycomplex) << endl;
cout << "Imaginary: " << imag(mycomplex) << endl;
return 0;
}
OUTPUT
Real:      10
Imaginary: 12
```

16

16

Declaration of variables

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

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

17

17

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.

18

18

Example

```
// test on global and local variables
#include <iostream>
using namespace std;
void f12(void);
int nglobal = 1;
Int 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;
}
```

```
main 1: nglobal = 1
main 2: nglobal = 2
f12   : nglobal = 2
main 3: nglobal = 3
```

19

19

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);
```

Initialization 3:

identifier = initial value ;

```
sum = 0.0;
```

20

20

Constants

Declared constants (most common for C++)

const type identifier = initial_value ;

Constant variable can not be modified thereafter.

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

Define constants (most common for C)

#define identifier value

```
#define PI 3.14159265
```

21

21

Example

```
//declaration of variables (example)
#include <iostream>
using namespace std;
#define PI 3.1415926
const float Ry = 13.6058;
int main()
{
    float a, speed, force_12;
    int   i, n;
    float angle = 45.0;
    ... some operators ...
    return 0;
}
```

22

22

Part 3: Operators

23

Operators

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

24

24

Assignment operator (=)

The assignment operator assigns a value to a variable.

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

```
a = 12 b = 12
```

25

25

Arithmetic operators

There are five arithmetic operators

Operator	Symbol C++	example
1. addition	+	f + 7
2. subtraction	-	p - c
3. multiplication	*	b * k
4. division	/	x / y
5. modulus	%	r % s

26

26

Precedence of arithmetic operators

Operator	Operation	Order
()	Parentheses	Evaluated first
*, / ,%	Multiplication	Evaluated second
	Division	(if more than one
	Modulus	then left-to-right)
+, -	Addition	Evaluated last
	Subtraction	(if more than one
		then left-to-right)

Example: a + b*c; step 1: b*c step 2: a + the result from step 1

Example: (a+b)*c; step 1: a+b step 2: c*(result from step 1)

27

27

Arithmetic assignment operators

There are five arithmetic assignment operators

Operator	C++	explanation
+=	a += 7	a = a + 7
-=	b -= 4	b = b - 4
*=	c *= 5	c = c * 5
/=	d /= 3	d = d / 3
%=	e %= 9	e = e % 9

However, you may find it's more explanatory to write a=a+7 than a+=7!

28

28

The increment/decrement operators

Operator	called	C++	
++	pre increment	++a	(add 1)
++	post increment	a++	
--	pre decrement	--a	(subtract 1)
--	post decrement	a–	

`x = x+1;` is the same as `x++;`

It seems there is not much value to use ++ or --.

29

29

Equality and relational operators

Equality operators in decision making

	C++	example	meaning
=	==	x == y	x is equal to y
	!=	x != y	x is not equal to y

Relational operators in decision making

	C++	example	meaning
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
	>=	x >= y	x is greater or equal to y
	<=	x <= y	x is less than or equal to y

30

30

Logical operators

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

Operator	Symbols	C++ example
and	&&	if (i==1 && j>=10)
or	||	if (speed>=10.0 || t <=2.0)

31

Conditional 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

```
// 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 << endl;
   return 0;
}
```

c=7

32

Part 4: Basic Input/Output

33

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>	contains cin, cout, cerr, clog.
<iomanip>	information for formatting
<fstream>	for file processing

34

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)

```
// output
#include <iostream>
using namespace std;
int main ()
{
int a;
   a=2;
    cout << " a = " << a << endl;
return 0;
}
OUTPUT
a=2
```


35

Example

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

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


36

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

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


Output for n = 2 and t = 4.0
   2    4.0000
```

37

37

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.

38

38

Example

```
  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;


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

39

39

Part 5: Control structures

40

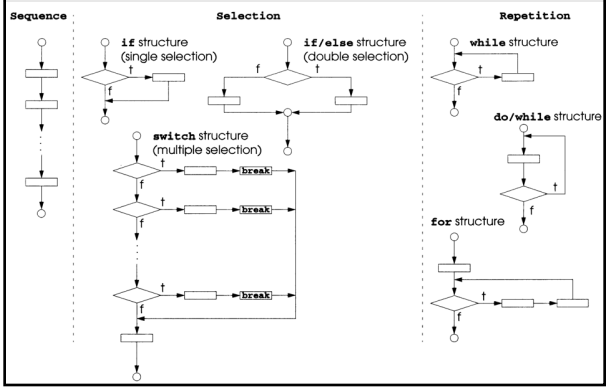
Control Structures

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.

41

Sequence, Selection, and Repetition



42

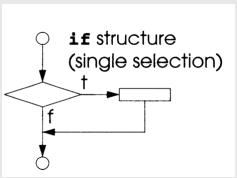
Three types of selection structures:

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

43

if - single-selection structure

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



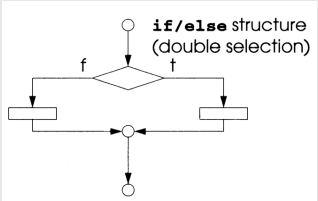

```
if (grade >=60)
  cout << "passed";

if (grade >=60) {
  n=n+1;
  cout << "passed";}
```

44

if/else - double-selection structure

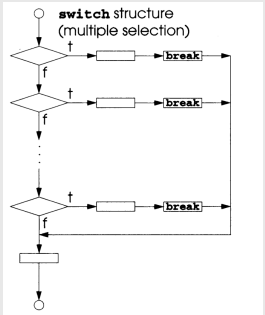
The if/else selection structure allows the programmer to specify that a different action is to be performed when the condition is true than when the condition is false.




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

45

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";
}
```

46

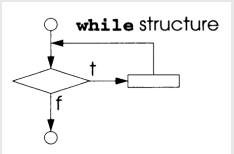
Three types of repetition structures:

1. while
2. do/while
3. for

47

The while repetition structure

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

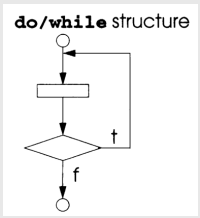



```
int n = 2;
while (n <= 100)
  {n = 2 * n;
   cout < n;}
```

48

The do/while repetition structure

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



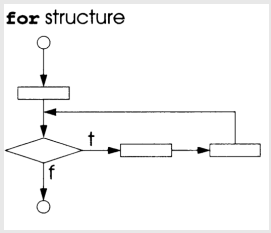

```
do {
   statement
} while (condition);
```

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

49

The for repetition structure

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




```
for (i=0; i <=5; i=i+1)
{
 … actions …
}
```

50

The break and continue statements

The break and continue statements alter the flow of the control.

The break statement, when executed in a while, for, do/while, or switch structure, causes 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.

51

```
// using the break statement
#include <iostream>
using namespace std;
int main ()
{
int n;
for (n = 1; n <=10; n = n+1)
{
   if (n == 5)
      break;
   cout << n << "  ";
}
cout << "\nBroke out of loop at n of " << n << endl;
return 0;
}
```

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

52

```
// using the continue statement
#include <iostream>
using namespace std;
int main()
{
    for ( int x=1; x<=10; x++)
    {
        if (x == 5)
        {continue;}
        cout << x << " ";
    }
    cout << "\nUsed continue to skip printing 5" << endl;
    return 0;
}
```

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

53

Good practice:

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

54

Part 6: Functions

55

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*.

56

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)    pow(x,y)
fabs(x)     floor(x)    fmod(x,y)   ceil(x)
```

57

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	
`<math.h>`		`<cmath>`	math library
`<iostream.h>`		`<iostream>`	input and output
`<fstream.h>`		`<fstream>`	read and write (disk)
`<stdlib.h>`		`<cstdlib>`	utility functions

… and many more

58

examples

old style

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

new style (note – add a line)

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
using namespace std;
```

59

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.

60

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.

61

```
//example: a programmer-defined function
#include <iostream>
using namespace std;
int square( int );   // function prototype
int main()
{
   for ( int x = 1; x <= 10; x++ )
   cout << square( x ) << "  ";
   cout << endl;
   return 0;
}
// Function definition
int square( int y )
{
   int result;
   result = y * y;
   return result;
}
```

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

62

Functions definitions

If a function does not receive any values *parameter-list* is **void** or left empty. If a function does not return any value, then return-value-type of that function is **void** both in the function prototype and function definition

```
//example: a "void" case
#include <iostream>
using namespace std;
void out2(void);   // function prototype
int main()
{
   out2();
   return 0;
}
// Function definition
void out2(void)
{
    cout << "output from function out2" << endl;
   return;
}
```

```
OUTPUT
output from function out2
```

63

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 this 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.

64

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.

```
// call-by-reference
#include <iostream>
using namespace std;
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;
}
```

```
OUTPUT
a = 12  b = 12
a = 24  b = 27
```

65

Default Arguments

Function calls may pass a particular value of an argument. The programmer can specify that such an argument is a *default argument* with a default value.

When a default argument is omitted in a function call, the default value is automatically inserted by the compiler and passed in the call.

Default argument must be the rightmost arguments in a function's parameter list.

Default arguments normally are specified in the prototype

```
int function2(int a=2);
```

66

Part 7: Arrays

67

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 every array is the 0^{th} element.

68

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"

69

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};
```

70

```
// Initialize array a and fill with numbers
#include <iostream>
#include <iomanip>
using namespace std;
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;
}
```

```
OUTPUT
Element      Value
      0          0
      1          2
      2          4
      3          6
      4          8
```

71

Multidimensional Arrays

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

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

	0	1	2	3	4
0	5	4	6	0	6
1	2	1	4	6	3
2	5	7	4	21	0

toys [2] [3] = 21;

72

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**.

```
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.

73

```
// Passing Arrays to Functions
#include <iostream>
using namespace std;
void print_array (int [], int);
int main ()
{
       int a[] = {1, 2, 3, 4};
       int b[] = {5, 4, 3, 2, 1};
       print_array (a,4);
       print_array (b,5);
       return 0;
}
void print_array (int arg[], int length)
{
       for (int n=0; n<length; n=n+1)
       cout << arg[n] << " ";
       cout << "\n";
}
```

```
OUTPUT
1 2 3 4
5 4 3 2 1
```

74

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.

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

75

```
// Static and Dynamic arrays
#include <iostream>
using namespace std;
void print_array (int [], int);
int main ()
{
       int a[5];
       static int b[5];
       print_array (a,5);
       print_array (b,5);
       return 0;
}
void print_array (int arg[], int length)
{
       for (int n=0; n<length; n=n+1)
       cout << arg[n] << " ";
       cout << "\n";
}
```

```
2147340288 4328756 1 256 1
0 0 0 0 0
```

76

Part 8: Input/Output with files

77

File processing (open and write)

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

Open a file with a name "file1.dat" and write to it

```
#include <iostream>
#include <fstream>
using namespace std;
ofstream outfile ("file1.dat", ios::out);
…
outfile << a << endl;
```

78

File processing (more)

Example 2 (also works)

Open a file with a name "file2.dat" and write to it

```
#include <iostream>
#include <fstream>
using namespace std;
ofstream outfile;

outfile.open("file2.dat");

outfile << a << endl;
```

79

File processing (open and read)

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

```
#include <iostream>
#include <fstream>
using namespace std;
ifstream inputfile ("input.dat", ios::in);

inputfile >> a;
```

To close a file

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

80

File open modes

Mode	Description
ios::app	Write all output to the end
ios::in	Open a file for input
ios::out	Open a file for output
ios::nocreate	If the file does not exist, the open operation fails
ios::noreplace	If the file exists, the open operation fails

81

Part 9: Pointers

82

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.

83

Declarations

Pointer variables contain memory address as their values.

Declaration:

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

84

Pointer operations

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

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

the statement `yptr = &y;`

assigns the address of the variable **y** to pointer **yptr**

Now the statement `cout << *yptr << endl;`

print the value of **y**, namely **5**.

And the statement `*yptr = 9;`

would assign **9** to **y**.

85

```
// Cube a variable using call-by-reference
// with a pointer argument
#include <iostream>
using namespace std;
void cubeByReference( int * );   // prototype
int main()
{
   int number = 5;
   cout << "The side is " << number;
   cubeByReference( &number );
   cout <<"\nThe volume is "<< number << endl;
   return 0;
}
void cubeByReference( int *nPtr )
{
 *nPtr = *nPtr * *nPtr * *nPtr;//cube to main
}
```

```
The side is 5
The volume is 125
```

86

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.

87

```
//example: using function pointers
#include <iostream>
using namespace std;
float av( float, float, float(*)(float));
float x2(float);
int main()
{ float x2average, xmin, xmax;
    xmin = 2.0;
    xmax = 4.0;
    x2average = av(xmin, xmax, x2);
    cout << "average = " << x2average << endl;
    return 0;
}

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

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

```
average = 10
```

88

Part 10: Examples

89

```
// Example 1: calculate values of a function
//            and write to a file
#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
    a = 0.0;             //left endpoint
    b = 2.0*pi;          //right endpoint
    n = 12;              //number of points
```

see the next slide ...

90

```
    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;
}
```

see the next slide …

91

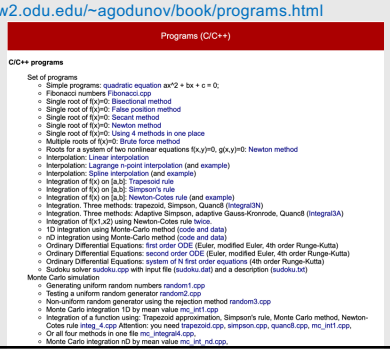
```
// Function f(x)
   double f(double x)
{
    double y;
    y = sin(x);
return y;
}
```

```
     x           f(x)
     0.00000     0.00000
     0.57120     0.54064
     1.14240     0.90963
     1.71360     0.98982
     2.28479     0.75575
     2.85599     0.28173
     3.42719    -0.28173
     3.99839    -0.75575
     4.56959    -0.98982
     5.14079    -0.90963
     5.71199    -0.54064
     6.28319    -0.00000
```

92

Examples

Many examples can be found on
https://ww2.odu.edu/~agodunov/book/programs.html



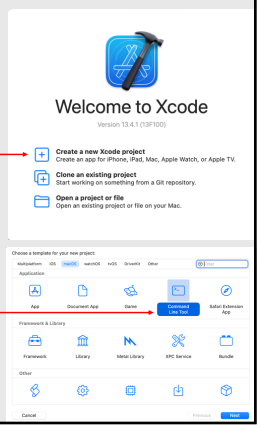
93

Part 11: Running C++ on macOS

94

Using Xcode on macOS

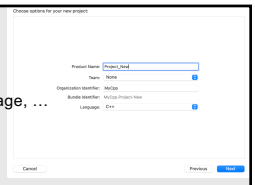
1. Download and install Xcode
2. Launch Xcode and double click on "Create a new Xcode project"
3. Choose "Command line tool" for macOS



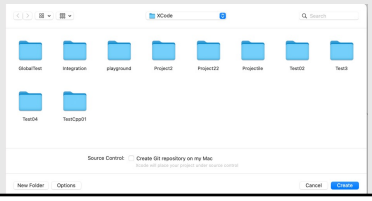
95

Using Xcode on macOS

4. Choose options: Project name, language, …



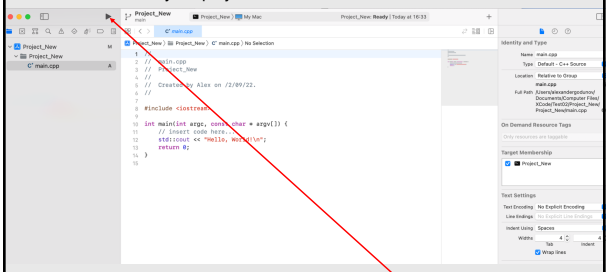
5. Choose (or create) a folder for your project



96

Using Xcode on macOS

6. The view of your project with Xcode

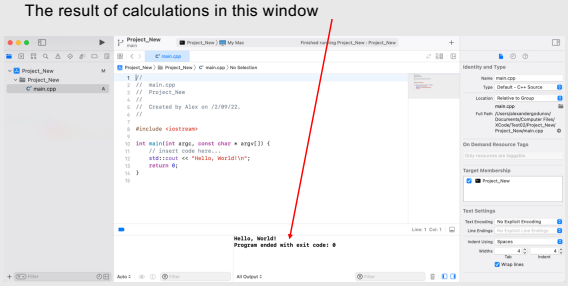


7. Compile and rune you code by clicking here
(the result is on the next slide …)

97

Using Xcode on macOS

The result of calculations in this window



98

Using command line on macOS

1. Launch the terminal application (one of macOS tools)
2. Navigate to your file you want to compile and run using command
   ls shows lists all files in the directory
   cd change the current working directory to a specific folder
   e.g. cd Project2
3. Run the compiler as
   g++ -o a.out project2.cpp
   note: a.out is the name of the executable file and project2.cpp is the C++ file (you can compile more than one file)
4. Type
   ./a.out
   to run the executable file

99

Using command line on macOS

For editing .cpp files you can use

- Xcode editor
- TextEdit
- or any other editor

100