ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στον Προγραμματισμό
Introduction to Programming

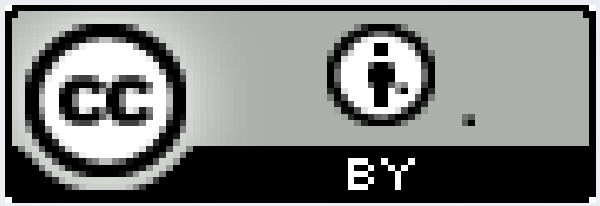
Τελική Επισκόπηση

Γ. Παπαγιαννάκης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται στην άδεια χρήσης **Creative Commons** και ειδικότερα

Αναφορά Δημιουργού 3.0 - Μη εισαγόμενο Ελλάδα (Attribution 3.0– Unported GR)



- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ΗΥ-150 Προγραμματισμός
CS-150 Programming

Lecture final:
A course review

G. Papagiannakis

The Aims

- Teach/learn
  - Fundamental programming concepts
  - Key useful techniques
  - Basic Standard C++ facilities
- After the course, you'll be able to
  - Write small colloquial C++ programs
  - Read much larger programs
  - Learn the basics of many other languages by yourself
  - Proceed with an "advanced" C++ programming course
- After the course, you will ***not*** (yet) be
  - An expert programmer
  - A C++ language expert
  - An expert user of advanced libraries

Lecture 1: (Chapter 1-2) Programming

- Why C++?
- Why software?
- Where is C++ used?
- Hello World program
- Computation & Linking
- What is programming?
- Integrated Development Environment

```
#include "std_lib_facilities.h " //header

int main() // where a C++ programs start
{
    cout << "Hello, world\n";// output
    keep_window_open();  // wait
    return 0;          // return success
}
```

Lecture 2: (Chapter 3) Types

- Builtin types: int, double, bool, char
- Library types: string, complex
- Input and output
- Operators—"overloading"
- Variable names in C++
- Simple computations
- Literals
- Declaration & initialization
- Type safety
- Programming philosophy

```
// inch to cm and cm to inch conversion:
int main()
{
    const double cm_per_inch = 2.54;
    int val;
    char unit;
    while (cin >> val >> unit) {// keep reading
        if (unit == 'i')      // 'i' for inch
            cout << val << "in == "
                 << val*cm_per_inch << "cm\n";
        else if (unit == 'c')          // 'c' for cm
            cout << val << "cm == "
                 << val/cm_per_inch << "in\n";
        else
            return 0; // terminate on a "bad
                                  // unit", e.g. 'q'
    }
}
```

Lecture 3: (Chapter 4) Computation

- Expressing computations
  - Correctly, simply, efficiently
  - Divide and conquer
  - Use abstractions
  - Organizing data, **vector**
- Language features
  - Expressions
    - Boolean operators (e.g. ||)
    - Short cut operators (e.g. +=)
  - Statements
  - Control flow
  - Functions
- Algorithms

```
// Eliminate the duplicate words; copying unique words
        vector<string> words;
string s;
while (cin>>s && s!= "quit")
        words.push_back(s);
sort(words.begin(), words.end());
vector<string>w2;
if (0<words.size()) {
        w2.push_back(words[0]);
        for (int i=1; i<words.size(); ++i)
                if(words[i-1]!=words[i])
                  w2.push_back(words[i]);
 }
cout<< "found " << words.size()-w2.size()
        <<  " duplicates\n";
for (int i=0; i<w2.size(); ++i)
         cout << w2[i] << "\n";
```

Lecture 4: (Chapter 5) Errors

- Errors ("bugs") are unavoidable in programming
  - Sources of errors?
  - Kinds of errors?
- Minimize errors
  - Organize code and data
  - Debugging
  - Testing
- Do error checking and produce reasonable messages
  - Input data validation
  - Function arguments
  - Pre/post conditions
- Exceptions– **error()**

```
int main()
{
    try
    {
        // ...
    }
    catch (out_of_range&) {
    cerr << "oops – some vector "
    " index out of  range\n";
    }
    catch (…) {
        cerr << "oops – some exception\n";
    }
    return 0;
}
```

Lecture 5: (Chapter 6) Writing a Program

- Program a simple desk calculator
  - Process of repeatedly analyzing, designing, and implementing
- Strategy: start small and continually improve the code
- Use pseudo coding
- Leverage prior work
  - Expression Grammar
  - Functions for parsing
- Token type
- Program organization
  - Who calls who?
- Importance of feedback

```
double primary()  // Num or ‘(‘ Expr ‘)’
{
    Token t = get_token();
    switch (t.kind) {
    case '(':                                   //
    handle ‘(’expression ‘)’
    {       double d = expression();
            t = get_token();
            if (t.kind != ')') error("')'  expected");
            return d;
    }
    case '8': // ‘8’ represents number “kind”
            return t.value; // return value
    default:
            error("primary expected");
    }
}
```

Lecture 6: (Chapter 7) Completing a Program

- Token type definition
  - Data members
  - Constructors
- Token_stream type definition
  - Function members
  - Streams concept
- "Grow" functionality: eg. prompts, and error recovery
- Eliminate "magic" constants

```
class Token_stream {
    bool full;        // is there a Token in the buffer?
    Token buffer; // here is where we keep a Token
public:
    Token get();      // get a Token
    void putback(Token); // put back a Token
    // the buffer starts empty:
    Token_stream() :full(false), buffer(0) { }
};

void Token_stream::putback(Token t)
{
    if (full) error("putback() into a full buffer");
    buffer=t;
    full=true;
}
```

Lecture 7: (Chapter 8) Functions

- Declarations and definitions
- Headers and the preprocessor
- Scope
  - Global, class, local, statement
- Functions
- Call
  - by value,
  - by reference, and
  - by **const** reference
- Namespaces
  - Qualification with :: and **using**

```
namespace Jack {//  in Jack's header file
        class Glob{ /*...*/ };
        class Widget{ /*...*/ };
}


#include "jack.h";// this is in your code
#include "jill.h";    // so is this


void my_func(Jack::Widget p)
{       // OK, Jack's Widget class will not
        // clash with a different Widget
        // …
}
```

Lecture 8: (Chapter 9) Classes

- User defined types
  - **class** and **struct**
  - **private** and **public** members
    - Interface
  - **const** members
  - constructors/destructor
  - operator overloading
  - Helper functions
  - Enumerations **enum**
- **Date** type

```
// simple Date (use Month type)
class Date {
public:
    enum Month {
            jan=1, feb, mar, apr, may, jun, jul,
            aug, sep, oct, nov, dec
    };
    Date(int y, Month m, int d);  // check for valid
                                  // date and initialize
    // …
private:
    int y;              // year
    Month m;
    int d;              // day
};

Date my_birthday(1950, 30, Date::dec); // error:
                                       // 2nd argument not a Month
```

Lecture 9: (Chapter 10) Streams

- Devices, device drivers, libraries, our code
- The stream model,
  - type safety, buffering
  - operators << and >>
- File types
  - Opening for input/output
  - Error handling
    - check the stream state
- Code logically separate actions as individual functions
- Parameterize functions
- Defining >> for **Date** type

```
struct Reading {// a temperature reading
        int hour; // hour after midnight [0:23]
        double temperature;
        Reading(int h, double t) :hour(h), temperature(t) { }
};

string name;
cin >> name;
ifstream ist(name.c_str());
vector<Reading> temps;  // vector of readings
int hour;
double temperature;
while (ist >> hour >> temperature) { // read
        if (hour < 0 || 23 <hour)
                error("hour out of range");
        temps.push_back( Reading(hour,temperature) ); // store
}
```

Lecture 10: (Chapter 11) Customizing I/O

- Formatted output—manipulators for **int** and **double**
- File open modes
- Text vs binary files
- Positioning in a filestream
- **stringstream***s*
- Line and **char** input/output
- Character classification functions

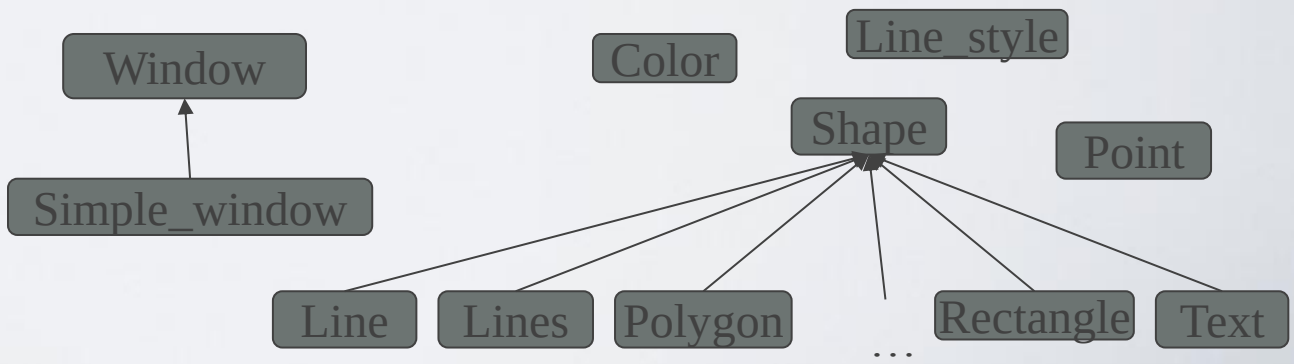
```
double str_to_double(string s)
// if possible, convert characters
// in s to floating-point value
{
    istringstream is(s);        // make a stream double d;
    is >> d;
    if (!is) error("double format error");
    return d;
}

double d1 = str_to_double("12.4"); // testing
double d2 = str_to_double("1.34e-3");
// will call error():
double d3 = str_to_double("twelve point four");
```

Lecture 11: (Chapter 12) Graphics

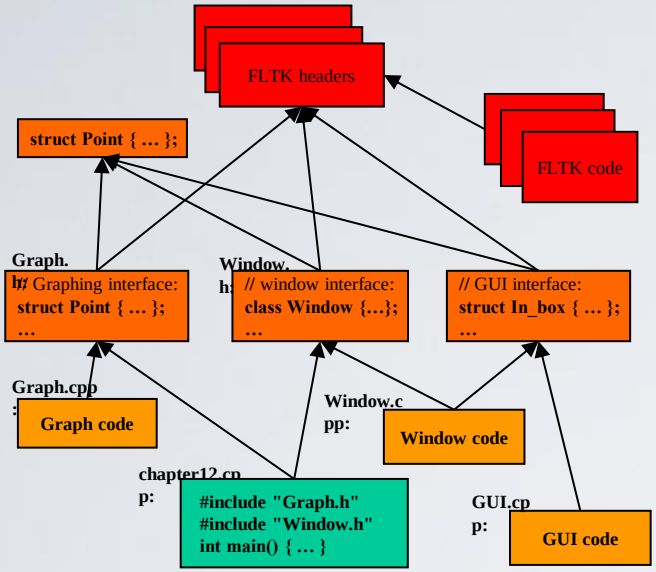
- Why Graphics/GUI?
- WYSIWYG
- Display model
  - Create a Window
  - Create Shapes
  - Attach objects
  - Draw
- 2D Graphics/GUI library
  - FLTK
  - Layered architecture
  - Interface classes

```
int main()
{
    using namespace Graph_lib; // use graph library
    Point tl(100,200);          // a point (obviously)
    Simple_window win(tl,600,400,"Canvas"); Polygon poly;  // make a polygon shape
    poly.add(Point(300,200));// add three points
    poly.add(Point(350,100));
    poly.add(Point(400,200));
    poly.set_color(Color::red);          // make it red
    win.attach(poly);     // connect poly to the window
    win.wait_for_button(); // give up control
}
```



Lecture 12: (Chapter 13) Graphics Classes

- Code organization




```
Simple_window win20(pt,600,400,"16*16 color  matrix");


Vector_ref<Rectangle> vr;  // use like vector

   // but imagine that it holds references to objects

for (int i = 0; i<16; ++i) { // i is the horizontal
                             // coordinate

   for (int j = 0; j<16; ++j) { // j is the vertical
                                // coordinate

      vr.push_back(

            new Rectangle(  Point(i*20,j*20),20,20)

            );

      vr[vr.size()-1].set_fill_color(i*16+j);

      win20.attach(vr[vr.size()-1]);

   }
```

- Implementation of **Point**, **Line,Color**, **Line_style**, **Polylines**, **Text**, etc.
- Object-oriented programming

Lecture 13: (Chapter 14) Design Principles for Programming a Class Library

- Implement types used in the application domain
- Derived classes inherit from a few key abstractions
- Provide a minimum number of operations, access functions
- Use a consistent, regular style, appropriate naming
- Expose the interface only
  - encapsulation
- Virtual functions
  - dynamic dispatching

```
void Shape::draw() const
        // The real heart of class Shape
        // called by Window (only)
{
        Fl_Color oldc = fl_color();          // save old color
            // there is no good portable way of
            // retrieving the current style (sigh!)
        fl_color(line_color.as_int()); // set color and
                                                // style
        fl_line_style(ls.style(),ls.width());

        // call the appropriate draw_lines():
        draw_lines();  // a "virtual call"
                                // here is what is specific for a
                                // "derived class" is done

        fl_color(oldc);          // reset color to previous
        fl_line_style(0);        // (re)set style to default
}
```

Lecture 14: (Chapter 16) GUI

- Graphical I/O
- Layered architecture
- Control inversion
  - Callbacks
  - Wait loops
  - Event oriented actions
- Buttons
- Input/output boxes

```
Button start_button(Point(20,20), 100, 20,
    "START", cb_start);
…
static void cb_start(Address, Address addr) {
    reference_to<Window>(addr).start();
}
void start(void) { start_pushed = true; }
….
void wait_for_start(void){
    while (!start_pushed) Fl::wait();
    start_pushed = false;
    Fl::redraw();
}
….
Window win (Point(10,10), “My Window");
….
win.wait_for_start();
```

Lecture 15: (Chapter 17) Free Store

- Built vector type
- Pointer type
- The **new** operator to allocate objects on the free store (heap)
- Why use free store?
- Run-time memory organization
- Array indexing
- Memory leaks
- **void***
- Pointers vs references

```
class vector {
    int sz;                 // the size
    double* elem;           // a pointer to the elements
public:
    // constructor (allocate elements):
    vector(int s) :sz(s), elem(new double[s]) { }
    // destructor (deallocate elements):
    ~vector() { delete[ ] elem; }
    // read access:
    double get(int n) { return elem[n]; }
    // write access:
    void set(int n, double v) { elem[n]=v; }
    // the current size:
    int size() const { return sz; }
};
vector v(10);
for (int i=0; i<v.size(); ++i) {
    v.set(i,i); cout << v.get(i) << ' ';
 }
```

Lecture 16: (Chapter 18) Arrays

- Vector copy constructor
- Vector copy assignment
- Shallow and deep copy
- Arrays—avoid if possible
- Overloading [ ]
  - i.e. defining [] for **vector**

```
class vector {
    int sz;                 // the size
    double* elem;           // pointer to elements
public:
    // constructor:
    vector(int s) :sz(s), elem(new double[s]) { }
    // …
    // read and write access: return a reference:
    double& operator[ ](int n) { return elem[n]; }
};

vector v(10);
for (int i=0; i<v.size(); ++i) {      // works and
                                      // looks right!
    v[i] = i;
// v[i] returns a
// reference to the ith element
    cout << v[i];
}
```

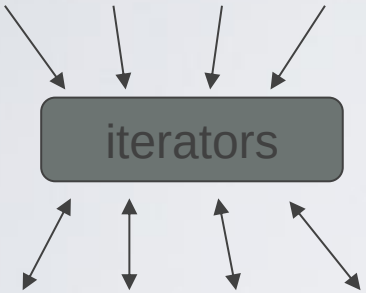
Lecture 17: (Chapter 19) Vector

- Changing vector size
- Representation changed to include free *space*
- Added
  - **reserve(int n)**,
  - **resize(int n)**,
  - **push_back(double d)**
- The *this* pointer
- Optimized copy assignment
- Templates
- Range checking
- Exception handling

```
// an almost real vector of Ts:
template<class T> class vector { // "for all types T" int sz;
                // the size
    T* elem;            // a pointer to the elements
    int space;          // size+free_space
public:
    // default constructor:
    vector() : sz(0), elem(0), space(0);
    // constructor:
    explicit vector(int s)
            :sz(s), elem(new T[s]), space(s) {
     // copy constructor:
    vector(const vector&);
    // copy assignment:
    vector& operator=(const vector&);
    ~vector() { delete[ ] elem; }    // destructor
    // access: return reference
    T& operator[ ] (int n) { return elem[n]; }
    int size() const { return sz; } // the current size
```

Lecture 18: (Chapter 20) The STL

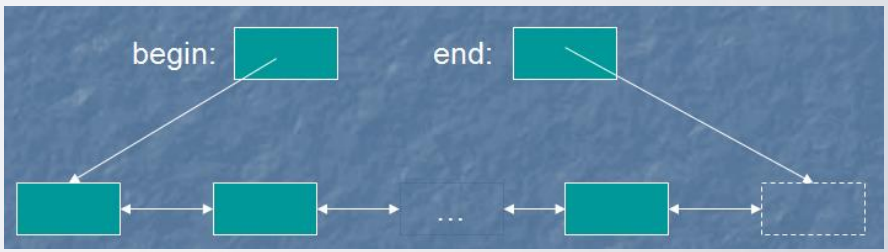
- Generic programming
  - "lifting an algorithm"
- Standard Template Library
- 60 Algorithms
  - sort, find, search, copy, …



  - vector, list, map, hash_map,…
- 10 Containers
- iterators define a sequence
- Function objects

```
// Concrete STL-style code  for a more
// general version of summing values

template<class Iter, class T>      // Iter should be an
                // Input_iterator
                                   // T should be
                // something we can                    // + and =
T sum(Iter first, Iter last, T s)  // T is the
                // "accumulator type"
{
    while (first!=last) {
            s = s + *first;
            ++first;
    }
    return s;
}
```



Lecture 19: Algorithms

- Associative containers
  - map, set
- Standard algorithms
  - copy, sort, …
  - Input iterators and output iterators
- List of useful facilities
  - Headers, algorithms, containers, function objects

- An STL-style algorithm
  - Takes one or more sequences
    - Usually as pairs of iterators
  - Takes one or more operations
    - Usually as function objects
    - Ordinary functions also work
  - Usually reports “failure” by returning the end of a sequence

Lecture 21: Text & Numerics

- Strings
- I/O
- Precision, overflow, sizes, errors
- Matrices
- Random numbers
- Complex numbers
- examples

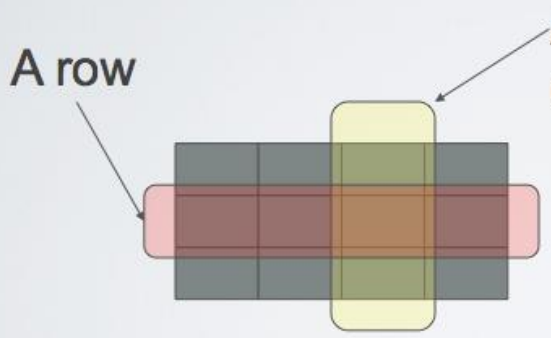
- Simple to_string

```
template<class T> string     to_string(const T& t)
{
        ostringstream os;
        os << t;
        return os.str();
}
```

- For example:

```
string s1 = to_string(12.333);
string s2 = to_string(1+5*6-99/7);
```

- **Matrix<int> m(3,4);**



A 3*4 matrix,
also called a 2 dimensional Matrix

3 rows
4 columns

Example of previous exam topics I

- Υλοποιήστε ένα πρόγραμμα το οποίο δέχεται από την γραμμή εντολών το σύμβολο μιας πράξης (‘+’, ’-‘ ,’*’ ,’/’), στην συνέχεια τους δυο αριθμητικούς όρους της και εκτυπώνει το αποτέλεσμά της πράξης αυτής. Διαβάστε την πράξη σε ένα string και με την χρήση if statements διαλέξτε την πράξη που επιθυμεί ο χρήστης και πραγματοποιήστε την.

Solution I

```
int main()
try
{
        string operation;
        double val1 = 0; // first/leftmost operand
        double val2 = 0; // second/rightmost operand
        cout << "Please enter an operation (+, -, *, /, plus, minus, mul, div) followed by
two floating-point values separated by a space: ";

        while(cin >> operation >> val1 >> val2) {
                double res = 0;
                if (operation=="plus" || operation=="+") res = val1+val2;
                else if (operation=="minus" || operation=="-") res = val1-val2;
                else if (operation=="mul" || operation=="*") res = val1*val2;
                else if (operation=="div" || operation=="/") {
                        if (val2==0) error("trying to divide by zero");
                        res = val1/val2;
                }
                else error("sorry: bad operator: ",operation);

                cout << val1 << operation << val2 << " == " << res <<'\n';
                cout << "Please try again: ";
        }
        cout << "exit because of bad input\n";
        keep_window_open("~");    // For some Windows(tm) setups
}
catch (runtime_error e) {
        cout << e.what() << '\n';
        keep_window_open("~");    // For some Windows(tm) setups
}
```

Example of previous exam topics II

- Υλοποιήστε ένα πρόγραμμα που διαβάζει από τον χρήστη μια γραμμή κειμένου και την εκτυπώνει ανάποδα χρησιμοποιώντας:
  - Αναδρομή σε συνδυασμό με iterators

Solution II

```
void printBackward( string::reverse_iterator,
   string::reverse_iterator ); // prototype

int main()
{
   string s;

   cout << "Enter a string: ";
   getline( cin, s );

   if ( !s.empty() )
   {
      // reverse_iterator r points
      // one location beyond the end of the reverse string
      string::reverse_iterator r = s.rend();

      // call recursive function printBackward
      printBackward( s.rbegin(), r - 1 );
      cout << endl;
   }
} // end main

// function to print the reverse string
void printBackward( const string::reverse_iterator s,
   string::reverse_iterator rb )
{
   // if the end is reached, return
   if ( rb == s - 1 )
      return;

   // recursive call to go through the string
   printBackward( s, rb - 1 );
   cout << *rb;
} // end function printBackward
```

Acknowledgements

Bjarne Stroustrup

Programming -- Principles and Practice Using C++

http://www.stroustrup.com/Programming/

Thank you!


BY


Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης