


Programming in C/C++

Lecture 4

<http://cs.vu.nl/~nsilvis/C++/2009>

Coordinator: Dr. Natalia Silvis-Cividjian (nsilvis@few.vu.nl)
Lectures: Dr. Jeffrey Napper



vrije Universiteit amsterdam

Questions?

- Lectures?
- Assignments?

Programming in C/C++ / Lecture 4 / 23-02-2009 2

Outline

- Generic programming
 - How to do less work yourself
- Standard Template Library (STL)
 - How to use other programmers' work
- 2D data visualization
 - How to see the results of your work

Programming in C/C++ / Lecture 4 / 23-02-2009 3

Generic programming

- Definition
- Function templates
- Operator overloading
- Class templates

Programming in C/C++ / Lecture 4 / 23-02-2009 4

Generic programming

- Goal is to write code that is independent of data types
 - Allows programmer to concentrate on algorithms
 - Allows code reuse across different types
 - Code can be provided in standard libraries!
- **Templates** are C++ tools for creating generic programs
 - Similar (but not exactly the same as) Java **generic**

Programming in C/C++ / Lecture 4 / 23-02-2009 5

Templates

- Function templates: for algorithm abstraction
 - Define a function independent of the data type T on which the function operates
 - `Swap<T>(a,b)`
- Class templates: for data abstraction
 - Define a class (an object) independent of the data type T of members—often used for *containers*
 - `Set<T>`, `List<T>`, `Stack<T>`, ...

Programming in C/C++ / Lecture 4 / 23-02-2009 6

Generic programming

- Definition
- **Function templates**

Programming in C/C++ / Lecture 4 / 23-02-2009

7

Problem: generic algorithms

- Define max(l,r) for every type?
 - Waste of space (*code bloat*): every definition is almost identical
 - COPY and PASTE may lead to errors and very bored programmers

```
// returns the maximum
// of 2 integers
int max(int left,
        int right)
{
    if (left < right)
        return right ;
    else
        return left ;
}

// returns the maximum
// of 2 doubles
double max(double left,
           double right)
{
    if (left < right)
        return right ;
    else
        return left ;
}
```

Programming in C/C++ / Lecture 4 / 23-02-2009

8

What do we really want?

- Programmer should define general function definition which works for both int and double parameters (and others!)
 - Generic functions can be supplied in a library
- Compiler should generate the code for each data type
 - Compiler doesn't get bored
 - Compiler doesn't make copy-paste errors
 - Compiler can generate code for data types as needed---can reduce code bloat

Programming in C/C++ / Lecture 4 / 23-02-2009

9

Solution: Function templates

- A *function template* is a pattern used by the compiler to automatically generate a family of function definitions
 - This is a powerful feature of C++
- To define a function template, use:

```
template<class T> T func(T arg)
```
- T will be replaced by the compiler with the data type for which the function code makes sense
 - The compiler will complain if it cannot!

Programming in C/C++ / Lecture 4 / 23-02-2009

10

Function templates: example

- A function template for the function max:

```
template <class T>
T myMax (T left, T right)
{
    if (left < right)
        return right ;
    else
        return left ;
}
```
- Here "class T" means "type T"
- In myMax(), the compiler must compare "<"
 - Hence, T can be any type for which the operator "<" is defined

Programming in C/C++ / Lecture 4 / 23-02-2009

11

Function templates: example

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

.. place here the function template

int main()
{
    int integer1 = 4;
    int integer2 = 10;

    // Note: compiler can figure out "int" type here!
    int max1 = myMax(integer1, integer2);
    cout << "The maximum integer is " << max1 << endl;

    double double1 = 100.20;
    double double2 = 5.7;

    double max2 = myMax<double>(double1, double2);
    cout << "The maximum double is " << max2 << endl;

    return 0;
}
```

```
template <class T>
T myMax (T left, T right)
{
    if (left < right)
        return right ;
    else
        return left ;
}
```

Programming in C/C++ / Lecture 4 / 23-02-2009

12

[Example output]

- The output is:

```
The maximum integer is : 10
The maximum double is : 100.20
```

- **Note:** some older compilers do not accept separate compilation for function templates
 - In this case function templates should be in the same file with main()---not in a header file!

Programming in C/C++ / Lecture 4 / 23-02-2009

13

[Template error example]

- `myMax(integer1, double1)`
 - error: no matching function for call to 'myMax(int&, double&)'
 - Compiler cannot infer type of T because both arguments are supposed to be T!
- `myMax<int>(integer1, double1)`
 - warning: passing 'double' for argument 2 to 'T maxb(T, T) [with T = int]'
 - Compiler knows type, but "double1" is wrong

Programming in C/C++ / Lecture 4 / 23-02-2009

14

[Generic programming]

- Definition
- Function templates
- **Operator overloading**

Programming in C/C++ / Lecture 4 / 23-02-2009

15

[Why can't "T" be any type?]

- **Remember:** T can be ANY type for which the code in the function definition makes sense (the compiler doesn't complain!)
- In `myMax()`, T can be any type for which the operator "<" is defined: int, double, float, char (only lower or only upper case)
 - What if we want to compare 2 arrays or 2 structs?
 - Write your own "<" operator for other types!

Programming in C/C++ / Lecture 4 / 23-02-2009

16

[Operator overloading]

- An *overloaded* operator has more than one definition
 - Can implement different algorithms for different types
 - `A + B` for matrices, etc.
 - **Not** allowed in Java (for reasons discussed later)

Programming in C/C++ / Lecture 4 / 23-02-2009

17

[Operator overloading rules]

- All operators can be overloaded except:
 - `.` (member access from an object)
 - `::` (scope resolution)
 - `.*` (member object selector, don't ask)
 - `?:` (if-then-else expression)
- You cannot change the unary/binary nature of an operator
- You cannot override *precedence* rules

Programming in C/C++ / Lecture 4 / 23-02-2009

18

Operator overloading: example "<<"

- The operator "<<" is used for output basic type variables:
`cout << a << 'a' << 5.0/2`
- How can we use "<<" to print larger variables (arrays, structures)?
 - Solution: overload the "<<" binary operator

Programming in C/C++ / Lecture 4 / 23-02-2009

19

Ex. 1: Overloading operator<<

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

struct Student
{
    string name;
    int stud_nr;
    int grade;
};

// Overload operator << to output a structure
// of type Student with newline
ostream& operator<< (ostream& outs,
                    const Student& the_object)
{
    outs << the_object.name << " "
        << the_object.stud_nr << " "
        << the_object.grade << endl;
    return outs; // allow chaining "<<"!
}
```

Programming in C/C++ / Lecture 4 / 23-02-2009

20

Ex. 1: Overloading operator<<

```
int main()
{
    Student me, you;
    me.name = "Sheila";
    me.stud_nr = 14537780;
    me.grade = 8;

    you.name = "Bob";
    you.stud_nr = 14532180;
    you.grade = 4;

    cout << me << you;

    return 0;
}
```

Output:
Sheila 14537780 8
Bob 14532180 4

Programming in C/C++ / Lecture 4 / 23-02-2009

21

Notes on operator<<()

- By default operator "<<" is bitwise left-shift
 - `ostream` class overloads this operator by providing a function with the prototype:
`ostream& operator<< (ostream&, type)`
- Any function that overloads "<<" has to return a reference to the `ostream` object parameter
 - This allows: `cout << "a chain "` "<<" of "<<" "output"
- `ostream` can be `cout` or any other stream (including files)

Operator overloading: example "<"

- The operator "<" is defined only for basic types (`int`, `real`, `char`)
- How can we compare two structs?
 - Solution: overload the "<" operator
 - Then, we can use `myMax()` for other types

Programming in C/C++ / Lecture 4 / 23-02-2009

23

Ex. 2: Overloading operator<

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

struct Student
{
    string name;
    int stud_nr;
    int grade;
};

// overloads "<" operator to compare 2 structs of type Student
bool operator< (const Student& student1, const Student& student2)
{
    return (student1.grade < student2.grade);
}
```

Ex. 2: Overloading operator<

```
int main()
{
    Student me, you;
    me.name = "Sheila";
    me.stud_nr = 14537780;
    me.grade = 8;
    you.name = "Bob";
    you.stud_nr = 14532180;
    you.grade = 4;

    if (me < you)
        cout << you.name << " is more clever than "
              << me.name << endl;
    else
        cout << me.name << " is more clever than "
              << you.name << endl;

    return 0;
}
```

Programming in C/C++ / Lecture 4 / 23-02-2009

25

Why not overload an operator?

- Can be wrong
 - `int operator+(int a, int b) { return a - b; }`
- Can be confusing
 - What order does student A < B use?
 - This is why Java doesn't allow operator overloading
- In general, overload operator<< for printing
 - Use more descriptive function names instead
 - Will need to overload some operators to use libraries

Programming in C/C++ / Lecture 4 / 23-02-2009

26

Generic programming

- Definition
- Function templates
- Operator overloading
- **Class templates**

Programming in C/C++ / Lecture 4 / 23-02-2009

27

Problem: generic classes

<pre>// A class for a list // of integers struct Node { int data; Node* next; }; class List { public: List(); ~List(); void addNode (int newdata); friend ostream& operator<<(ostream& outs, List& list_to_print); private: Node *head; };</pre>	<pre>// A class for a list // of doubles struct Node { double data; Node* next; }; class List { public: List(); ~List(); void addNode (double newdata); friend ostream& operator<<(ostream& outs, List& list_to_print); private: Node *head; };</pre>
---	--

Programming in C/C++ / Lecture 4 / 23-02-2009

28

What do we really want?

- Programmer should define general List class for lists of any type (doubles, char, classes)
 - Generic classes can be provided by a library
- Compiler should generate type-specific code
 - As with function templates, compiler should do all the work!

Programming in C/C++ / Lecture 4 / 23-02-2009

29

Solution: class templates

- A *class template* is a pattern used by the compiler to automatically generate class types
- To define a class template use:

```
template<class T> class myType { }
```
- A *container* is a generic class
 - Examples: lists, sets, queues, etc.
 - See also: `java.util.*`

Programming in C/C++ / Lecture 4 / 23-02-2009

30

Class templates: example

```
// Generic Node
// structure
template<class T>
struct Node
{
    Node<T>* next;
    T data;
};

// Generic List container
template<class T>
class GenericList
{
public:
    void add(const T& data);
    List();
    ~List();

    friend ostream& operator<< (
        ostream& outs,
        GenericList<T>& list);
private:
    Node<T> *head;
};
```

Programming in C/C++ / Lecture 4 / 23-02-2009

31

Class templates: example

```
... implementation file is not shown here....
int main()
{
    // Note: compiler needs to know type here!
    GenericList<int> first_list ;
    first_list.add(1) ;
    first_list.add(2) ;
    cout << first_list ;
    GenericList<char> second_list ;
    second_list.add ('A') ;
    secondlist.add('B');
    cout << second_list ;
    return 0 ;
}
```

Outline

- Generic programming
- **Standard Template Library (STL)**

Programming in C/C++ / Lecture 4 / 23-02-2009

33

Standard Template Library (STL)

"Don't reinvent the wheel; use libraries"

B. Stroustrup, The programming C++ language, 2000

- STL is a large collection of templates for useful classes and functions
 - Developed by A. Stepanov and M. Lee at Hewlett Packard Labs in 1994
- STL is tested, debugged, offers high performance, and is free
 - Practically, STL now is part of C++ language
 - Similar to the way `java.util.*` is part of Java

Programming in C/C++ / Lecture 4 / 23-02-2009

34

More information

- <http://www.sgi.com/tech/stl/>
- Read chapter 16 in Prata's book

Programming in C/C++ / Lecture 4 / 23-02-2009

35

STL overview

- STL contains:
 - generic classes (class templates, or containers)
 - generic algorithms (function templates)
- Examples:
 - Containers: class templates for vector, list, stack, queue, map, set
 - Iterators: pointer-like objects to access container elements
 - Algorithms: function templates for operations on containers

Programming in C/C++ / Lecture 4 / 23-02-2009

36

STL containers

- Containers contain data of a single type and functions to operate on that data
- Some frequently used STL containers:
 - vector: for dynamic arrays (order by index)
 - list: for doubly-linked list
 - stack: for LIFO (last in first out) structures
 - queue: for FIFO (first in first out) structures
 - deque: for double-ended queue
 - priority_queue: for order by max value
 - map: for key-value pairs

Programming in C++ / Lecture 4 / 23-02-2009

37

Vectors: #include <vector>

- Vectors are like arrays that can grow and shrink while your program is running
 - Elements of the vector can be of any type.
- Example:

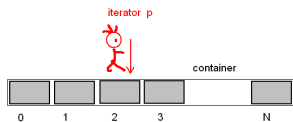
```
#include <vector>
using namespace std;
// declare a vector of integers
vector<int> my_vector ;
// declare a vector of objects of type my_Class
vector<my_Class> my_class_vector ;
```

Programming in C++ / Lecture 4 / 23-02-2009

38

How do you access elements?

- Each container type has its own *iterator* type to access its data
 - vector<int>::iterator iterV ;
 - list<double>::iterator iterL ;



Programming in C++ / Lecture 4 / 23-02-2009

39

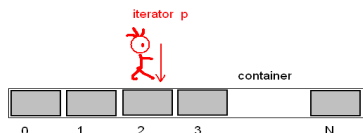
Using iterators

- Iterators are similar to pointers:
 - iter++ : advances to the next element in the container
 - iter-- : goes back to the previous element
 - *iter : gives access to the data pointed by p .
 - iter[2] or *(iter+2) : point to the same element
 - == and != : compares for equality or difference
 - Note:** iterators use operator overloading!
- Each container has member functions begin() and end() to return an iterator to the first and the last element in the container

Programming in C++ / Lecture 4 / 23-02-2009

40

How to walk through a container c?



```
vector<int>::iterator p; // declare p
for(p = c.begin() ; // start at begin
    p != c.end() ; // stop at end
    p++) { // move to next
    do_stuff(*p); // use element
}
```

Programming in C++ / Lecture 4 / 23-02-2009

41

How to work with a vector?

- Use my_vector[i] to read and change a vector element that already has a value
 - Example: my_vector[2] = 5;
 - Note:** [] cannot be used for initialization!
- Use push_back() to add an element to the end of the vector
 - Example: my_vector.push_back (10)
- Use pop_back() to delete the last element

Programming in C++ / Lecture 4 / 23-02-2009

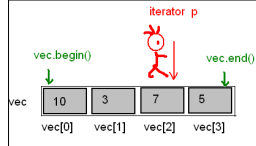
42

A vector container example

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

int main()
{
    vector<int> vec;
    vector<int>::iterator p;

    vec.push_back (10);
    vec.push_back (3);
    vec.push_back (7);
    vec.push_back (5);
    cout << "These are the contents of the vector: \n";
    for (p = vec.begin(); p != vec.end(); p++)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```



Programming in C/C++ / Lecture 4 / 23-02-2009

43

Lists are the same!

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

int main()
{
    list<int> myList;
    list<int>::iterator p;

    for (int i = 1; i <= 10; i++)
        myList.push_back(i*2);

    cout << "These are the contents of the list: \n";
    for (p = myList.begin(); p != myList.end(); p++)
        cout << *p << " ";
    cout << endl;
    return 0;
}
```

Programming in C/C++ / Lecture 4 / 23-02-2009

44

Generic algorithms

```
#include <algorithm>
```

- STL provides more than 70 standard generic algorithms for the most basic and commonly used operations of containers
- Each algorithm is implemented independently of the individual container type (that is, it is templated)

Programming in C/C++ / Lecture 4 / 23-02-2009

45

STL algorithms

- Some frequently used STL generic algorithms :
 - find(x) : finds x in a container
 - count(x) : counts the number of occurrences of x in the container
 - max_element() : returns the maximum element in a container
 - sort(b, e) : sorts a range of elements in a container
 - swap(a, b) : assigns the contents of a to b and the contents of b to a

Programming in C/C++ / Lecture 4 / 23-02-2009

46

Sorting a vector using STL

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

int main()
{
    vector<int> vec;
    vector<int>::iterator p;

    vec.push_back (10);
    vec.push_back (3);
    vec.push_back (7);
    vec.push_back (5);

    cout << "Vector before "
    << "sort:\n";

    for (p = vec.begin();
         p != vec.end();
         p++)
        cout << *p << " ";
}
```

```
cout << endl;
sort(vec.begin(), vec.end());

cout << "Vector after sort:\n";

for (p = vec.begin();
     p != vec.end();
     p++)
    cout << *p << " ";

cout << endl;
return 0;
}
```

Output: Vector before sort:
10 3 7 5
Vector after sort:
3 5 7 10

Programming in C/C++ / Lecture 4 / 23-02-2009

48

More information

- Prata, Appendix G
- S. Lipmann, C++ primer
- Internet

Outline

- Generic programming
- Standard Template Library (STL)
- **2D data visualization**

How to visualize the results?

- Write results to a file and:
 - use Microsoft Excel,
 - use gnuplot, or
 - pngwriter
- Many other options as well...

Microsoft Excel help

- Internet:
 - <http://atom.physics.calpoly.edu/Excel/excel.html>

GNUplot

- Data and mathematical function plotting program
- <http://www.gnuplot.info>

Gnuplot example:plots myplotdata.dat

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

int main()
{
    // First, write data as x,y pairs (ex: "1 4.5")
    // to "myplotdata.dat"...

    // To run gnuplot from your C++ program...
    system("d:/gnuplot/wgnuplot.exe d:/gnuplot/mycmdfile.txt");
    // or on kits (in directory with "mycmdfile.txt")...
    system("gnuplot mycmdfile.txt");

    return 0;
}
```

GNUplot command file

- Simple syntax to plot data as line graph:

```
set data style lines
plot 'd:\gnuplot\myplotdata.dat' title 'line 1'
pause -1
```

GNUplot help

- Use gnuplot help (in gnuplot program):

```
kits$ gnuplot
gnuplot> help
gnuplot> quit
```
- Internet:
 - <http://www.gnuplot.info/docs/node1.html>

Programming in C/C++ / Lecture 4 / 23-02-2009

55

PNGwriter

- Easy-to use open source graphics library that uses PNG (Portable Network Graphics) as output format
- The interface supports the following:
 - Plotting (no axes)
 - Basic shapes (lines, circles, rectangles)
 - Color (translucent, filled)
 - Scaling, interpolation

Programming in C/C++ / Lecture 4 / 23-02-2009

56

PNGwriter example

```
#include "pngwriter.h"

int main() {
    int i;
    pngwriter png(300, 300, // width and height
                 0, "test.png"); // background & filename

    // Loop over line to draw...
    for (i = 1; i < 300; i++) {
        double y = 150+100*sin((double)i*9/300.0);
        png.plot(i, y, // x,y coordinate of pixel
                0.0, 0.0, 1.0); // red,green,blue components
    }
    png.close();
    return 0;
}
```

Programming in C/C++ / Lecture 4 / 23-02-2009

57

PNGwriter help

- Internet:
 - <http://pngwriter.sourceforge.net/>
- Ask your grader for pointers to help

Programming in C/C++ / Lecture 4 / 23-02-2009

58

Self test exercises

1. Why should operator<< return the ostream argument?

```
template<class T>
ostream& operator<< (ostream& os, T obj) {
    return os;
}
```
2. Why must operator<< be declared a friend of the class?

Programming in C/C++ / Lecture 4 / 23-02-2009

59

Self test exercises

3. Declare a vector of type:

```
class Student {
    int grade;
    string name;
    int id;
}
```
4. Write a loop to find the student with a) the highest grade, then b) lowest id.
5. Can we use generic functions to find both?

Programming in C/C++ / Lecture 4 / 23-02-2009

60

[Self test exercises]



- 6. Why do we have to use `vector<int>` to declare a vector of `int`, but we can call `myMax(1, 29)` without specifying the `<int>`?
- 7. Why not use a template for every function?
- 8. When would you use a template instead of inheritance?

Programming in C/C++ / Lecture 4 / 23-02-2009

61

[Questions?]



- Thanks!
 - Good luck on the assignments

Programming in C/C++ / Lecture 4 / 23-02-2009

62