02393 Programming in C++ Module 13
Summary and Outlook

Teacher: Alberto Lluch Lafuente

Sebastian Mödersheim (slides author)

May 9, 2016

Lecture Plan

#	Date	Topic	Chapter *
1	1.2	Introduction	1
2	8.2	Basic C++	1
3	15.2	Data Types	2
4	22.2		
5	29.2	Libraries and Interfaces	3
6	7.3	Classes and Objects	4.1, 4.2 and 9.1, 9.2
7	14.3	Templates	4.1, 11.1
		Påskesferie	
8	4.4	Inheritance	14.3, 14.4, 14.5
9	11.4	Recursive Programming	5
10	18.4	Linked Lists	10.5
11	25.4	Trees	13
12	2.5	Graphs	16.1-16.3,16.5
13	9.5	Summary	
	17.5	Exam	

* Recall that the book uses sometimes ad-hoc libraries that are slightly different with respect to the standard libraries (e.g. strings and vectors).

The exam

- Date: 17 May 2016
- Time and place:
  - ★ 16:00-20:00 in 101/Hal 1
  - ★ 16:00-21:00 in 116/44
    (Students who have been granted extra time)

  check

  http://portalen.dtu.dk/DTU_Generelt/AUS/Studerende/Infosite/Lokaleoversigt_for_sommereksamen_2016.aspx
- Duration: 4 hours
- All written material permitted
- Marking: pass/fail
  - ★ 4 exercises of 2.5 points for a total of 10 points;
  - ★ You need more or less 5 points to pass the exam;
  - ★ If you did the CodeJudge assignments you have already some points.

The exam

Structure of the exercises:

- You will be given a main program `exZZ-main.cpp` and a library header file `exZZ-library.h`;
- The tasks are to check/correct/implement code in `exZZ-library.cpp`;
- Typically structured into tasks (a), (b), ... that you can solve incrementally. You get points for each task so that you do not need to check/correct/implement the whole library.
- The main program is like a test that you can use to check your solutions;
- The test can be run in *CodeJudge* as well.

The exam

Topics you can expect:

- Implement a recursive function;
- Iterate arrays/vectors/matrices/sets/etc.;
- Basic use of containers of the standard library like vectors, sets, maps, etc.;
- Implement class methods for a given class declaration, including constructors/destructors;
- Deal with pointers;
- Some tree-like structure.

The exam

Submission:

- Paper submission is allowed but not recommended;
- Electronic submission is through CampusNet;
- Submission through CodeJudge is mostly for yourself (to test your code);
- After submission, additional tests may be run on your code.

Arithmetics in C++

Consider this *code fragment*

```
if(( 0.1 + 0.2 ) - 0.3 == 0.0)
     cout << "YES" << n << endl;
else
     cout << "NO" << n << endl;
```

What would be the output?

Evaluation order in C++

What is the output of this?

```
bool even(unsigned int z){
    // Some implementation that tests if z is even
}

int main(void){
    int x = 0 ;
    int y = 1;

    if(x > y && even(x))
        cout << "yes" << endl;
    else
        cout << "no" << endl;

    if(even(x) && x > y)
        cout << "yes" << endl;
    else
        cout << "no" << endl;
}
```

Scope

Consider this *code fragment*

```
int n = 2;
int x = 0;
int i = 0;
{
    int x = 0;
    for(int i = 0; i <= n; i++){
        x = x + i;
    }
}
cout << "x=" << x << endl;
cout << "i=" << i << endl;
cout << "n=" << n << endl;
```

Do you forsee any compile-time error? Any run-time error?
What would be the output?

Side effects

Consider this *code fragment*

```
int x = 0;
int y = 0;
if (x = ((x++ - 1))) {
    y = 1;
}
if (x = ((++x - 1))) {
    y = 2;
}
cout << "y=" << y << endl;
cout << "x=" << x << endl;
```

Do you forsee any compile-time error? Any run-time error?
What would be the output?

Arrays

Consider this *code fragment*

```
int *b;
{
    int a[4];
    b = new int[4];
    for(int i = 0; i <= 4; i++){
        a[i] = b[i];
    }
}
```

Do you forsee any compile-time error? Any run-time error?
What would be the final contents of a and b?

STL (standard template library)

STL (standard template library): is a C++ library of container classes and algorithms

- a container class stores a collection of elements;
- manages the storage space for its elements;
- provides many of the very commonly used structures: sets, vectors, stacks, queues, dictionaries/maps, trees, etc.
- very important to know how to deal with them;
- may be implemented in different ways;

STL (standard template library)

Containers we have seen:

- vectors, sets and multisets (used and implemented);
- stacks and queues (used only in some examples)
- lists (briefly mentioned, and implemented)
- maps (briefly mentioned and used in some examples and solutions)

Suggestion: have a this link at hand in the exam

http://www.cplusplus.com/reference/stl/

and, possibly, have a look the briefly mentioned containers.

Arrays vs containers: iterating data

Iterating arrays

```
int a[n];
for(int i = 0; i < n; i++)
    f(a[i]);
```

Iteraring containers

```
vector<int> a(n);
for(int a = 0; i < a.length(); i++)
    f(a[i]);

for (iterator i = a.begin(); i != a.end(); i++)
    f(*i);

for (auto x : a)
    f(x);
```

Works for iterable containers (e.g. vector/set, but not stack/queue).

Type constructors

Type declarations

```
struct A {
    int x;
    A na;
};

struct B {
    int x;
    B * nb;
};
```

Variable declarations

```
A a;
B b;
A * c;
B * d;
```

Expressions

```
b.x
d.x
d->x
(*d).x
b.nb.x
b.nb->x
d->nb->nb->nb->x
```

Which type declarations are ok?
Which variable declarations are ok?
Which expressions are ok?

Recursive Data-Structures

Does this sound familiar?
Where have we used similar structures?

```
struct A {
    T k;
    A * p;
}

struct B {
    T k;
    B * p;
    B * q;
}
```

Pointers and arrays

Consider the following code fragment

```
int a[4] = {1, 2, 3, 4};

cout << a;
cout << *a;
cout << a[0];
cout << a[1];
cout << *(a) + 1;
cout << *(a+4);
```

Do you forsee any compile-time error? Any run-time error?
What would be the output?

Arrays as arguments

```
int f(int * a){
    int sum = 0;
    for(int i = 0; i < n; i++) sum += a[i]
}

int main(void){
    int a[4] = {1, 2, 3, 4};
    int sum = f(a);
    return 0;
}
```

Is this ok?

Pass by value, pass by reference

What is the output of this code fragment?

```
bool f(int x, int * y, int & z, int * & u){
    x = 0;
    * y = 0;
    z = 0;
    u = & z;
}

int main(void){
    int a = 1;
    int b = 2;
    int c = 3;
    int * p ;

    f(a,&b,c,p);
    cout << a << endl;
    cout << b << endl;
    cout << c << endl;
    cout << * p << endl;
}
```

Summary: Static vs. dynamic memory allocation

Static Allocation (on the stack)

- Created when entering the scope of the declaration (local variable or function parameter);
- Killed when the scope ends (e.g. function returns)

Dynamic Allocation (on the heap)

- Allocated with the new operator;
- Killed with `delete`;

Implementing a recursive function

The two-argument Ackermann function, is defined as follows for nonnegative integers m and n:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1,A(m,n-1)) & \text{otherwise} \end{cases}$$

Blahblah = *"In computability theory, the Ackermann function, named after Wilhelm Ackermann, is one of the simplest and earliest-discovered examples of a total computable function that..."*

Question: how do you implement this function in C++?

Hints: do not let the Blablah impress you. Focus on the definition. Choose appropriate types. Choose an appropriate type for your function. Use recursion to get an almost line-by-line transcription from math lines into C++ lines.

Implementing a recursive function

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{otherwise} \end{cases}$$

```
int A(int m, int n)
    if (m==0) return n +1;
    if (m>0 && n==0) return A(m-1,1);
    else return A(m-1, A(m, n-1));
}
```

Templates in C++

Consider this code fragment

```
template < typename T, int N >
T * f(void){
    T * p = new int[N];
    return p;
}

int main(void){
    int * a;
    int * b;
    int n;

    a = f<int,4>();

    cin >> n;
    b = f<int,n>();
}
```

What is f? What is going to happen?

Basic Inheritance

```
class A {
public: virtual void f(void) = 0;
};

class B : public A {
public: void f(void){ return; };
};

class C : public A {
public: void f(void){ return; };
        void g(void){ return; };
};

class D : public C { };

void foo(C c){
    return;
}
```

```
int main(void){
    A a;
    B b;
    C c;
    D d;

    b.f();
    b.g();
    d.g();
    foo(d);
}
```

Do you foresee any compile- or run-time error?

Inheritance: scopes

```
class A {
public:    int x;
protected: int y;
private:   int z;
};

class B : public A {
public:
    B(void){
        x = 1;
        y = 1;
        z = 1;
    }
};
```

```
class C : private B {
public:
    C(void){
        x = 2;
        y = 2;
        z = 2;
    }
};

class D : public C {
public:
    D(void){
        x = 3;
        y = 3;
        z = 3;
    }
};
```

Do you foresee any compile- or run-time error?

Public, Protected and Private Scopes/Inheritance

```
class A {
public: x; // accessible to everyone but E
protected: y; // accessible to all derived classes (A, B, C, D) but E
private: z; // accessible only to A
};

class B : public A {
    // x is public
    // y is protected
    // z is not accessible
};

class C : protected A {
    // x is protected
    // y is protected
    // z is not accessible
};

class D : private A {
    // x is private
    // y is private
    // z is not accessible
};

class E : public D {
    // x is not accessible
    // y is not accessible
    // z is not accessible
};
```

Inheritance: static and dynamic dispatch

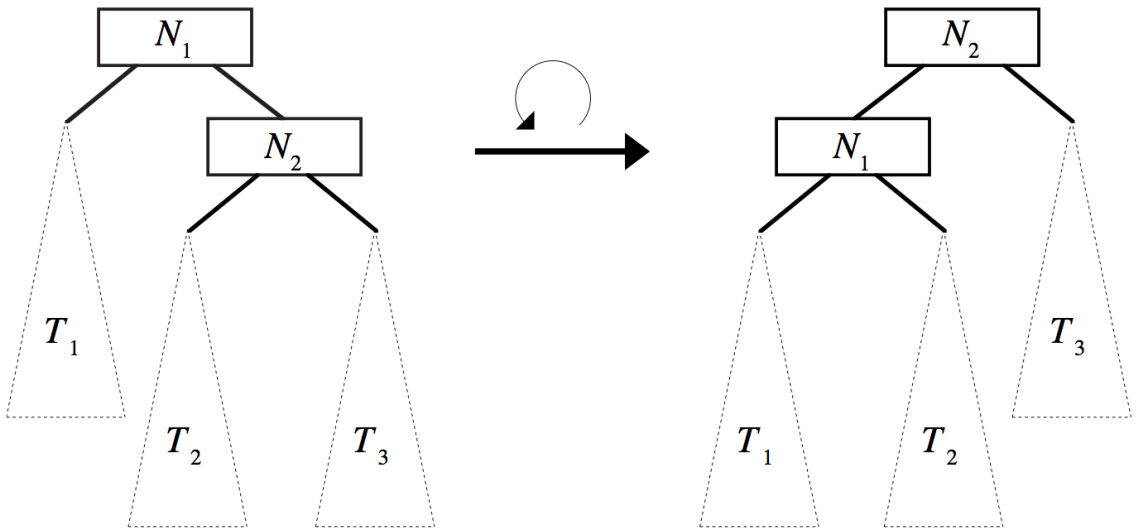
What is the output of this code fragment?

```
class Person {
public: virtual void h(void) { cout << "Hello" << endl; };
        void g(void) { cout << "Good bye" << endl; };
};

class Guy : public Person {
public: void h(void) { cout << "Hi" << endl; };
        void g(void) { cout << "Bye" << endl; };
};

int main(void){
    Guy bob;
    Person alice;
    Person * he;
    he = &alice;
    he->h();
    he->g();
    he = &bob;
    he->h();
    he->g();
}
```

Balancing trees: Rotations



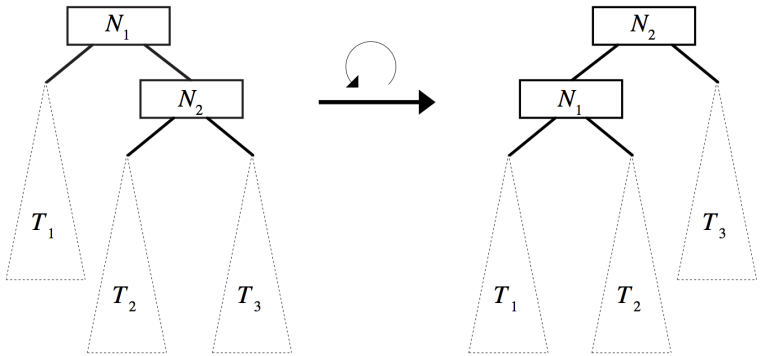
A balanced tree guarantees good performance of tree operations, e.g. $O(\log n)$ insertion/deletion/retrieval of elements.

However, inserting/deleting nodes may unbalance a tree. To restore them rotations can be applied.

Implementing rotations

Consider this structure for nodes in a tree

```
struct Node {
    T * data;
    Node * left;
    Node * right;
}
```



Is this a correct implementation of a right rotation?

```
Node * rotateLeft(Node * N1) {
    Node * N2;
    N2 = N1->right;
    N1->right = N2->left;
    N2->left = N1;
    T2->data = T1->data;
    T1->data = T2->data;
    return N2;
}
```

If not, how would you correct it?

Beyond this course

How to be a better (C++) programmer?

1. Practice, practice, practice;
2. Open your mind:
   - ★ Learn a new programming language/paradigms, e.g. *Functional Programming (in F#)* (02157, 02257);
   - ★ Understand the foundations of programming languages, e.g. *Computer Science Modelling* (02141);
3. Acquire programming skills, e.g.
   - ★ Algorithm design, e.g. *Algorithms & Data Structures* (02105, 02110)
   - ★ Code optimization, e.g. as done by *Compilers* (02247);
   - ★ Code analysis, as in techniques like *Program Analysis* (02242) and *Model Checking* (02246).

Beyond this course

How to be a better (C++) programmer of *reliable software*? [1]

DTU has a study line in "Reliable Software Systems" in that focuses on this. It includes courses like:

- *Compiler Construction* (02247): covers the basics of analysis and optimisation techniques applied during compilation.
- *Program Analysis* (02242) covers advanced analysis methods to spot errors and optimisations not catched by compilers.
- *Model Checking* (02246) focuses on errors of interacting software, e.g. to analyse that the software cannot get stuck.

+ secure systems (02244), embedded systems (02223), distributed systems (02220), cryptographic systems (02232), high-performance (02614), data processing (02632), ...

[1] The one that IT companies like Google, Microsoft and Intel deploy when they want to provide *rock-solid and performant* software-based services and products.