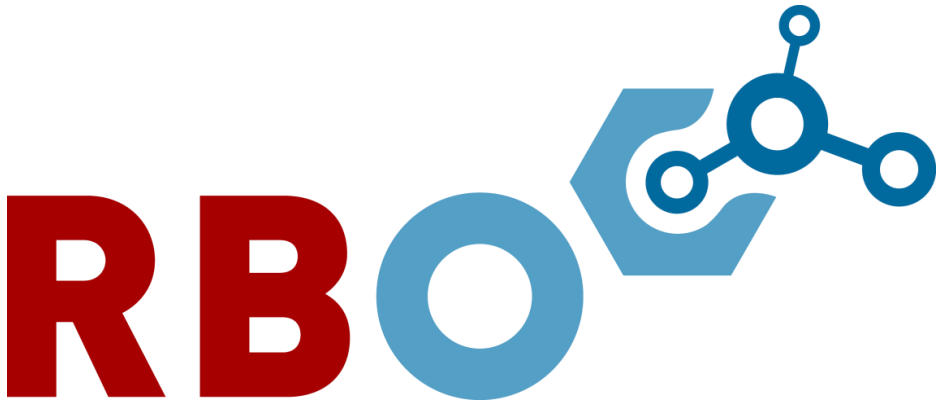


Intro to C++ For Programmers

Előd Páll

Technische Universität Berlin

TIOBE Index

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	↑	C#	3.399%	+0.10%
6	5	↓	Visual Basic .NET	3.291%	-2.02%
7	8	↑	JavaScript	2.128%	-0.00%
8	9	↑	SQL	1.944%	-0.12%
9	7	↓	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%
11	34	⇈	Groovy	1.502%	+1.20%
12	14	↑	Assembly language	1.378%	+0.15%
13	11	↓	Delphi/Object Pascal	1.335%	+0.04%
14	16	↑	Go	1.220%	+0.14%
15	12	↓	Ruby	1.211%	-0.08%
16	15	↓	Swift	1.100%	-0.12%
17	20	↑	Visual Basic	1.084%	+0.40%
18	13	⇊	MATLAB	1.062%	-0.21%
19	18	↓	R	1.049%	+0.03%
20	17	↓	Perl	1.049%	-0.02%

What is C++?

►Object-oriented programming language, used excessively for Linux and Windows

►Real superset of C

►Native = OS / architecture dependent

►Very powerful, but many complicated concepts
http://yosefk.com/c++fqa/

►"C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows your whole leg off."
Bjarne Stroustrup

Basics: Compiling & Building

- Compile & build:
  - `$ g++ helloworld.cpp -o helloworld`
  - `$ ./helloworld`
  - `makefile, Cmake, catkin_make`
    `(catkin build catkin-tools.readthedocs.io)`

- Debugging:
  - `$ g++ -g helloworld.cpp -o helloworld`
  - **`$ gdb`** `./helloworld`
  - On MacOS you might have to add ***`-save-temps`*** to g++

- Caveat:
  - You will usually not invoke the compiler yourself but let your IDE / Makefile do it. We will use different frameworks in later assignments and talk more about compiling and building

GNU Debugger Mini-Guide

- `r` (=run)
  - start the program; halts when error occurs
- `break helloworld.cpp:4` / `break 4`
  - halt when reaching line 4 in helloworld.cpp / current file
- `s` (=step)
  - step through the program line-wise (omitting function calls)
- `n` (=next)
  - step inside a function
- `p var` (=print)
  - print value of variable var
- `c` (continue)
  - continue execution
- `bt` (=backtrace)
  - inspect current function call stack
- `frame 3`
  - jump to frame 3 on stack trace
- `list`
  - list code in current frame

Hello World!

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

Hello World!

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    .
    .
    .
```

Using header files

Pointers and Arrays

Namespaces

Standard library

Basics: Variables, Loops, Functions, If-else

```
 1 // someFunction1.cpp
 2 #include <iostream>
 3 void someFunction(int n) {
 4     int result = 0;
 5     if (n <= 0) {
 6           result = 0;
 7     } else {
 8           for (int i = 1; i <= n; i++) {
 9                 result += i;
10           }
11     }
12     std::cout << result << std::endl;
13 }
14 int main( int argc, char* argv[] ) {
15     someFunction (10);
16     return 0;
17 }
```

Headers vs. Sources: Declaration vs. Definition

```
// someFunction2.h
int someFunction(int n);
```

```
// someFunction2_main.cpp
#include <iostream>
#include "someFunction2.h"

int main( int argc,
    char* argv[] ) {
  std::cout
    << someFunction (10)
    << std::endl;
  return 0;
}
```

```
// someFunction2.cpp
#include "someFunction2.h"

int someFunction(int n) {
  int result = 0;
  if (n <= 0) {
    result = 0;
  } else {
    for (int i=1; i <= n; i++) {
      result += i;
    }
  }
  return result;
}
```

```
$ g++ someFunction2.cpp someFunction2_main.cpp -o someFunction2
```

Namespaces

```
 1 // namespaces.cpp
 2 #include <iostream>
 3 namespace myns {
 4   int pow(int n) {
 5    return n*n;
 6   }
 7 }
 8
 9 int main( int argc, char** argv ) {
10    std::cout << myns::pow(5) << std::endl;
11  using namespace myns;
12  using namespace std;
13    cout << pow(5) << endl;
14
15    return 0;
16 }
```

Arrays

```
 1 // arrays.cpp
 2 #include <iostream>
 3 using namespace std;
 4
 5 int main( int argc, char* argv[] ) {
 6    int numbers[10]; // 'int[10] numbers' is WRONG
 7    numbers[0] = numbers[1] = 1;
 8    for (int i = 2; i < 10; i++) {
 9      numbers[i] = numbers[i-1] + numbers[i-2];
10    }
11
12    cout << numbers[2] << ", "
13     << numbers[3] << ", " << numbers[4] << endl;
14
15    return 0;
16 }
```

Arrays of variable size (1)

```
 1 // arrays_variable.cpp
 2 #include <iostream>
 3 #include <stdlib.h>
 4 int main( int argc, char* argv[] ) {
 5    if (argc < 2) return -1;
 6    int size = atoi(argv[1]);//load int from stdin
 7    int numbers[size]; // not allowed in C++!
 8    numbers[0] = numbers[1] = 1;
 9    for (int i = 2; i < size; i++) {
10      numbers[i] = numbers[i-1] + numbers[i-2];
11    }
12    std::cout << numbers[size-1] << std::endl;
13    return 0;
14 }
```

```
g++  -Wall -pedantic arrays_variable.cpp -o arrays_variable
./arrays_variable 20
```

Arrays of variable size (1)

```
 1 // arrays_variable.cpp
 2 #include <iostream>
 3 #include <stdlib.h>
 4 int main( int argc, char* argv[] ) {
 5    if (argc < 2) return -1;
 6    int size = atoi(argv[1]);//load int from stdin
 7    int* numbers = new int[size];
 8    numbers[0] = numbers[1] = 1;
 9    for (int i = 2; i < size; i++) {
10      numbers[i] = numbers[i-1] + numbers[i-2];
11    }
12    std::cout << numbers[size-1] << std::endl;
13    delete[] numbers;
14    return 0;
   }
```

The Dark Side (1): Stack vs. Heap

Stack

Static memory:
•Variables of size known at **compile time**
•Data is available within current scope `{ ... }`
•Managed by OS

```
int numbers[10];
```

Heap

Dynamic memory:
•Variables of size only known at **run time**
•Managed by user - always available
(memory leaks!)

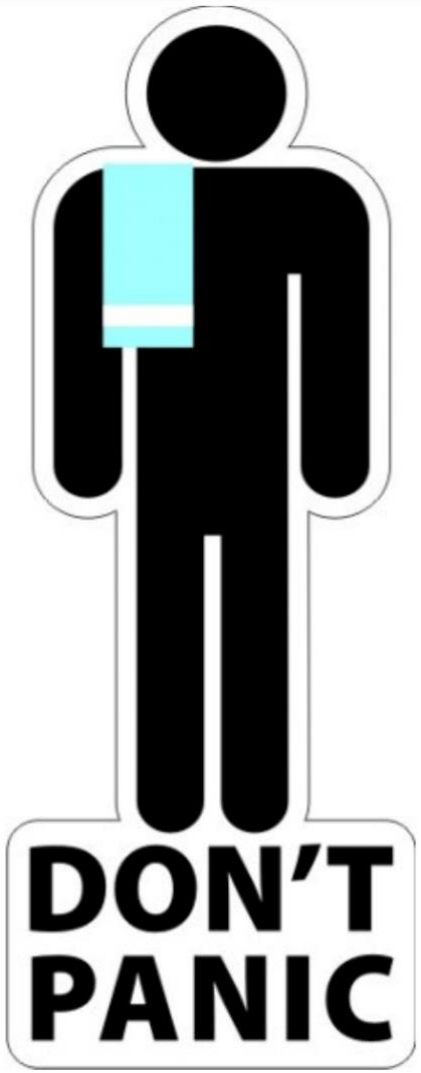
```
int* numbers
     = new int[size]
// ...
delete[] numbers;
```

The Dark Side (2): Pointers

```
 1 // darkside2.cpp
 2    // (int size) initialized from outside
 3    int* numbers = new int[size];
 4    numbers[0] = 10; numbers[2] = 5;
 5    cout << "A: " << numbers << endl;
 6    cout << "B: " << (*numbers) << endl;
 7    cout << "C: " << numbers[2] << endl;
 8    cout << "D: " << ((*numbers)+2) << endl;
 9    cout << "E: " << &(*numbers) << endl;
10    cout << "F: " << &size << endl;
```

```
A: 0x100200080          (Address of numbers on heap)
B: 10                   (Value of first int at address "numbers")
C: 5
D: 12
E: 0x100200080          (Address of numbers on heap)
F: 0x7fff5fbfec64       (Address of size on stack)
```

Don’t panic!



Classes: Declaration

```
 1 // IntList.h
 2 class IntList {
 3 protected: //private:
 4   // protected/private member variables
 5   int max_size;
 6   int* members;
 7   int current_size; //need to store array length
 8 public:
 9     IntList(int max_size_); //constructor
10     virtual ~IntList(); //destructor
11     // public member functions
12     bool add(int number);
13   // more members like elem(i)
14   // ...
15 };
```

Classes: Definition

```
1  // IntList.cpp
2  #include "IntList.h"
3  IntList::IntList(int max_size_)
4      : max_size(max_size_), current_size(0) {
5      members = new int[max_size];
6  }
7
8  IntList::~IntList() {
9      delete[] members;
10 }
11
12 bool IntList::add(int number) {
13     if (this->current_size+1 >= this->max_size) {
14       return false;
15     }
16     members[current_size] = number; // this-> is optional
17     current_size++;
18     return true;
19 }
20
```

Classes: Instantiating

```
 1  #include <iostream>
 2  #include "IntList.h"
 3
 4  int main( int argc, char* argv[] ) {
 5      IntList list(10); // declare AND init on STACK
 6      list.add(5);
 7      list.add(29);
 8      std::cout << list.elem(0) << std::endl;
 9
10      IntList* list2;              // declare pointer
11      list2 = new IntList(10);     // init on HEAP
12      (*list2).add(5);             // dereference pointer
13      list2->add(29);              // short notation
14      std::cout << list2->elem(0) << std::endl;
15
16      return 0;
17  }
```

```
(*pointerToObj).method()   =   pointerToClass->method()
```

Classes: Inheritance & Polymorphy

```
 1  // inheritance.cpp
 2  class A {
 3  public:
 4      virtual void alpha() {
 5        cout << "A:alpha" << endl;
 6      }
 7  };
 8  class B : public A {
 9  public:
10      void alpha() {
11        cout << "B:alpha" << endl;
12      }
13  };
14
15  int main( int argc, char* argv[] ) {
16      A *class1 = new A;
17      A *class2 = new B;
18      class1->alpha();
19      class2->alpha();
20      return 0;
21  }
```

Classes: Const Methods

```
 1 // const_methods.cpp
 2 class A {
 3     int alpha;
 4 public:
 5     virtual int getAlpha() const {
 6       //alpha = 1; // would give a compiler error!
 7       return alpha;
 8     }
 9     virtual void setAlpha(int a) {
10       alpha = a;
11     }
12 };
13
14 int main( int argc, char* argv[] ) {
15     A *a;
16     a = new A;
17     a->setAlpha(5);
18     cout << a->getAlpha() << endl;
19     return 0;
20 }
```

Introducing STL – The Standard Library (1)

```
 1 //stl.cpp
 2 #include <iostream>
 3 #include <vector>
 4 #include <string>
 5
 6 int main( int argc, char* argv[] ) {
 7    std::string name1("Klaus");
 8    std::string name2("Peter");
 9    std::vector<std::string> names;
10
11    names.push_back(name1);
12    names.push_back(name2);
13
14    std::cout << names[0] << std::endl;
15 }
```

Introducing STL – The Standard Library (2)

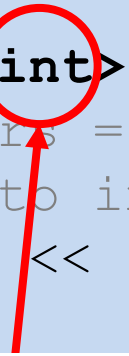
- Use `std::string` not `char*`
- Use `std::vector` not array

- STL implements many data structures (“containers”) and operations
  - Hash table (`std::map`)
  - Sorting (`#include <algorithm>`)
  - Tuples (`std::map`)
  - and many more

- Containers are *templated*

Introducing STL – The Standard Library (3)

```
1 // std2.cpp
2     std::vector<std::vector<int> > int2d;
3           // vector of vectors = 2d array!
4     // ...add some elements to int2d...
5     std::cout << int2d[0][1] << std::endl;
6
```



This space matters on Unix systems! (not >> operator)

Introducing STL – The Standard Library (4)

```
1  // std2.cpp
2     std::vector<std::vector<int> > int2d;
3           // vector of vectors = 2d array!
4     // ...add some elements to int2d...
5     std::cout << int2d[0][1] << std::endl;
6
7     std::vector<std::string> slist;
8     // ...add some elements to slist...
9
10    // using constant iterator to go through list
11    std::vector<std::string>::const_iterator it;
12    for (it=slist.begin(); it!=slist.end(); it++){
13      std::string current = *it;
14      std::cout << current << std::endl;
15    }
```

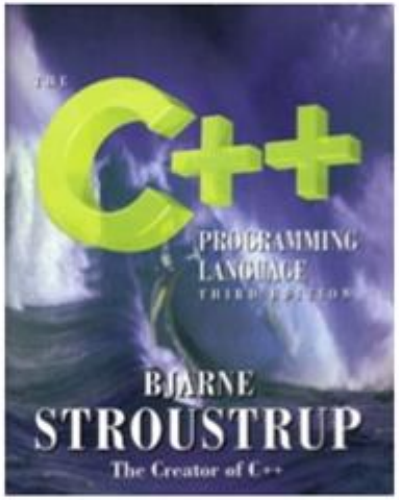
Call By Reference

```
1 // call_by_reference.cpp
2 struct Variables {//like class but all members public
3     double input;
4     double output;
5 };
6
```

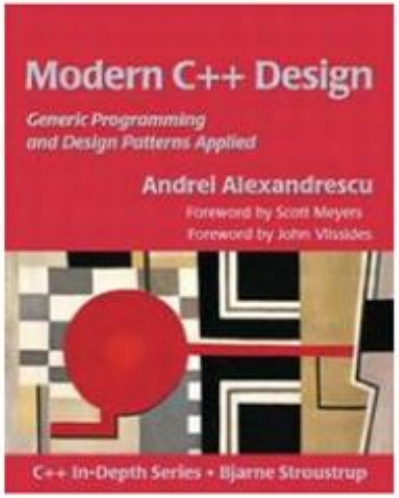
Fun with pointers

```
 1  // fun_with_pointers1.cpp
 2  #include <iostream>
 3  using namespace std;
 4
 5  struct MotorVariables {
 6      double output1;
 7      double output2;
 8  };
 9
10  MotorVariables* control(const double input) {
11      MotorVariables results;
12      results.output1 = 2.*input;
13      results.output2 = 4.*input;
14      return &results;
15  }
16
17  int main( int argc, char* argv[] ) {
18      MotorVariables* vars = NULL;
19      vars = control(10.0);
20      cout << "vars " << &vars << endl;
21      cout << vars->output1 << endl;
22  }
```

Books

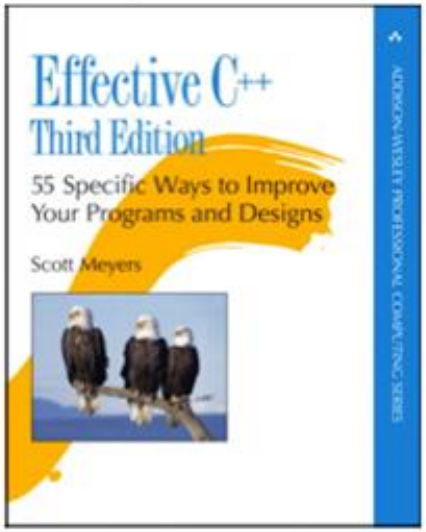


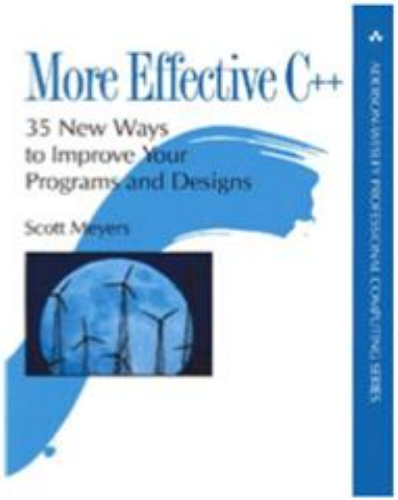
Bjarne Stroustrup:
The C++ Programming Language (3rd Edition)



Andrei Alexandrescu:
Modern C++ Design

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides ("Gang of Four" or GoF):
Design Patterns – Elements of Reusable Object-Oriented Software





Scott Meyers:
Effective C++
More Effective C++

Websites

►Google:
“Introduction to C++ for <your favorite lang> Programmers”

►Concise and free introduction:
http://www.learncpp.com/

►Online community for programmers:
https://stackoverflow.com/

More Tricky Concepts (Not Covered Here)

- ►Constants, const pointers, const methods
http://www.thomasstover.com/c_pointer_qualifiers.html
- ►References (safer concept than pointers)
- ►OOP: Inheritance, polymorphism, abstract classes...
- ►C++ Styles
http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml
- ►Boost: THE C++ library
http://www.boost.org
- ►Debugging C++: GNU Debugger (GDB)
http://www.sourceware.org/gdb/
- ►Building libraries: static and dynamic
http://www.learncpp.com/cpp-tutorial/a1-static-and-dynamic-libraries/
- ►Smart pointers (provide garbage collection to CPP)
- ►Function pointers

Brief, Incomplete and Mostly Wrong History of Programming Languages

►http://james-iry.blogspot.de/2009/05/brief-incomplete-and-mostly-wrong.html