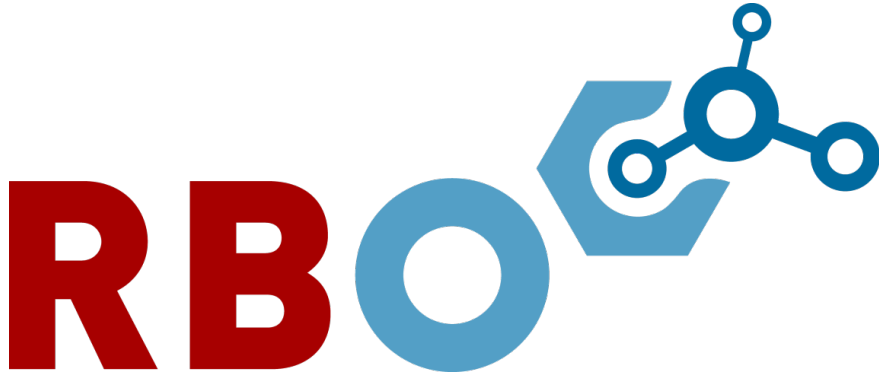


Intro to C++ for programmers

Aditya Bhatt

Technische Universität Berlin

C++ is popular

Oct 2020	Oct 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	16.95%	+0.77%
2	1	↓	Java	12.56%	-4.32%
3	3		Python	11.28%	+2.19%
4	4		C++	6.94%	+0.71%
5	5		C#	4.16%	+0.30%
6	6		Visual Basic	3.97%	+0.23%
7	7		JavaScript	2.14%	+0.06%
8	9	↑	PHP	2.09%	+0.18%
9	15	⇈	R	1.99%	+0.73%
10	8	↓	SQL	1.57%	-0.37%

https://www.tiobe.com/tiobe-index/

C++

- An Object-Oriented Programming (OOP) language
  - To know more about OOP: https://www.stroustrup.com/whatis.pdf
- A real superset of C
- Ideal for Systems Programming
  - complies to native code (OS/architecture dependent)
  - great for hardware-aware programming
- Powerful, with 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 and Building

Compiling and running programs

```
$ g++ helloworld.cpp -o helloworld
$ ./helloworld
```

Debugging

```
$ g++ -g helloworld.cpp -o helloworld
$ gdb ./helloworld
```

GNU Debugger

In the assignments, you will instead invoke a build system.

- `Makefile`
- `CMake`
- `catkin_make`

gdb cheatsheet

`r (=run)`
start the program; halt when an error occurs

`break helloworld.cpp:4` *or* `break 4`
halt when reaching line 4

`s (=step)`
step through the program line-wise (omitting function calls)

`n (=next)`
step inside a function

`p var (=print)`
print the value of a variable var

`c (=continue)`
continue program execution

`bt (=backtrace)`
inspect current function call stack

`frame 3`
jump to frame 3 on the stack trace

`list`
list code in the 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! – A Closer Look

```
#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;
    .
    .
    .
```

header files

pointers and arrays

namespaces

standard library

Basics: Variables, Loops, Functions

```
1  // someFunction1.cpp
2  #include <iostream>
3
4  void someFunction(int n) {
5
6      int result = 0;
7
8      if (n <= 0) {
9          result = 0;
10     }
11     else {
12
13         for (int i = 1; i <= n; i++) {
14             result += i;
15         }
16     std::cout << result << std::endl;
17 }
18
19 int main( int argc, char* argv[] ) {
20       someFunction (10);
21       return 0;
22 }
```

Declaration vs Definition

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

```
// 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 main.cpp -o someFunction2
```

Namespaces

```
1   // namespaces.cpp
2   #include <iostream>
3
4   namespace myns {
5     int pow(int n) {
6      return n*n;
7     }
8   }
9
10  int main( int argc, char** argv ) {
11      std::cout << myns::pow(5) << std::endl;
12
13      using namespace myns;
14      using namespace std;
15
16      cout << pow(5) << endl;
17       return 0;
18  }
```

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  }
```

Variable-sized Arrays – I

```
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
```

Variable-sized Arrays – II

```
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 – I : 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!)

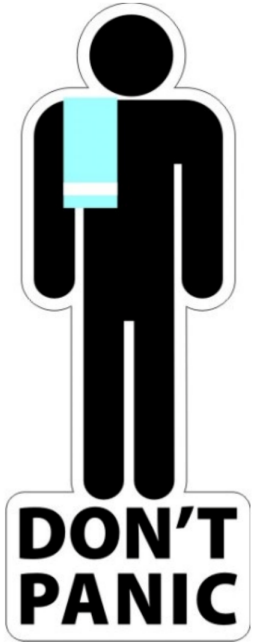
```
int* numbers
        = new int[size]
// ...
delete[] numbers;
```

The Dark Side – II : 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
B: 10
C: 5
D: 12
E: 0x100200080
F: 0x7fff5fbfec64
```

(Address of numbers on heap)
(Value of first int at address “numbers”)
(Address of numbers on heap)
(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
4  IntList::IntList(int max_size_)
5        : max_size(max_size_), current_size(0) {
6        members = new int[max_size];
7  }
8
9  IntList::~IntList() {
10       delete[] members;
11 }
12
13 bool IntList::add(int number) {
14       if (this->current_size+1 >= this->max_size) {
15         return false;
16       }
17       members[current_size] = number; // this-> is
18 optional
19       current_size++;
20       return true;
21 }
```

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 and 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  }
```

STL – The C++ Standard Library – I

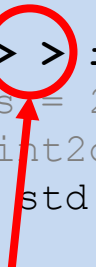
```
1  //stl.cpp
2  #include <iostream>
3  #include <vector>
4  #include <string>
5
6  int main( int argc, char* argv[] ) {
7        std::string name1("Klaus");
8        std::string name2("Peter");
9        std::vector<std::string> names;
10
11       names.push_back(name1);
12       names.push_back(name2);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

STL – The C++ Standard Library – II

- Use `std::string` not `char*`
- Use `std::vector` not array
- Many data structures (“containers”) and operations
  - Hash tables (std::map)
  - Sorting (#include <algorithm>)
  - Tuples
  - and many more
- Containers are *templated*

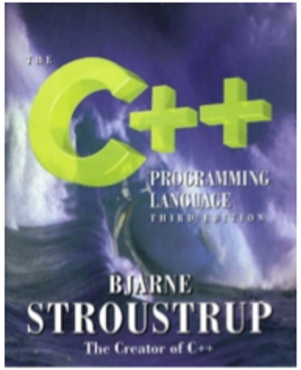
STL – The C++ Standard Library – III

```
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
```

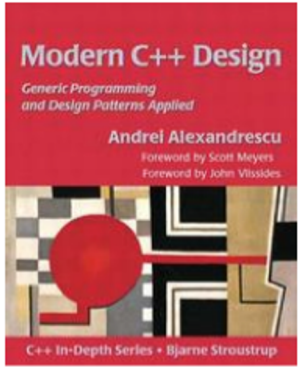


This space matters on Unix systems! (not >> operator)

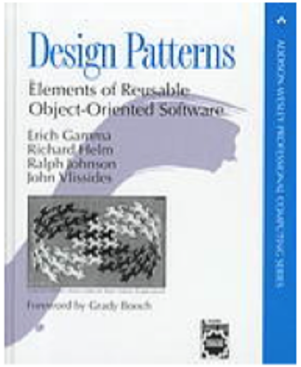
Books



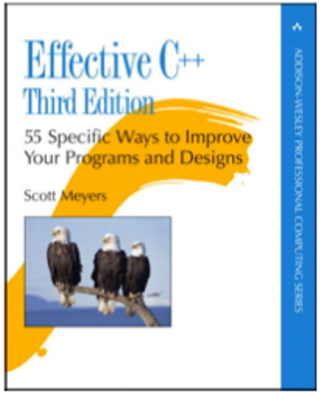
Bjarne Stroustrup:
The C++ Programming Language (3rd Edition)

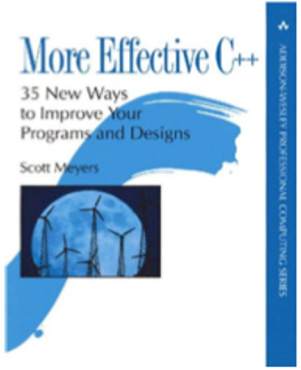


Andrei Alexandrescu:
Modern C++ Design



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++

More 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