Programming in C++ - Primer

Lesson 2 - Dive In

Jakub 'Eremiell' Marek
<marekj14@fel.cvut.cz>

Silicon Hill C++ Academy

2013/10/28

1 Revision

2 Formatting

3 Headers

4 I/O Streams

5 Variables

Welcome!

Revisiting basics

Revisiting basics

Conditions

Revisiting basics

Conditions

- if

Revisiting basics

Conditions

- if
- switch/case

Revisiting basics

Conditions

- if
- switch/case
- ternary

Revisiting basics

Conditions

- if
- switch/case
- ternary

Loops

Revisiting basics

Conditions

- if
- switch/case
- ternary

Loops

- while

Revisiting basics

Conditions

- if
- switch/case
- ternary

Loops

- while
- do/while

Revisiting basics

Conditions

- if
- switch/case
- ternary

Loops

- while
- do/while
- for

Formatting

Formatting

Compiler doesn't care about whitespace

Formatting

Compiler doesn't care about whitespace

- spaces

Formatting

Compiler doesn't care about whitespace

- spaces
- tabulators

Formatting

Compiler doesn't care about whitespace

- spaces
- tabulators
- newlines

Formatting

Compiler doesn't care about whitespace

- spaces
- tabulators
- newlines

You can format your code in many ways

Formatting

Compiler doesn't care about whitespace

- spaces
- tabulators
- newlines

You can format your code in many ways
Which can lead to unreadable code

Formatting

Compiler doesn't care about whitespace

- spaces
- tabulators
- newlines

You can format your code in many ways
Which can lead to unreadable code

The International Obfuscated C Code Contest

http://ioccc.org/

Formatting styles

Formatting styles

Allman (ANSI)

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

Formatting styles

Allman (ANSI)

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

Java

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Formatting styles

Allman (ANSI)

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

K&R

```
int Foo(bool isBar)
{
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Java

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Formatting styles

Allman (ANSI)

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

Java

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

K&R

```
int Foo(bool isBar)
{
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Stroustrup

```
int Foo(bool isBar)
{
     if (isBar) {
          bar();
          return 1;
     } else
          return 0;
}
```

Formatting styles

Allman (ANSI)

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

Java

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

K&R

```
int Foo(bool isBar)
{
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Stroustrup

```
int Foo(bool isBar)
{
     if (isBar) {
          bar();
          return 1;
     } else
          return 0;
}
```

Whitesmith

```
int Foo(bool isBar)
    {
    if (isBar)
        {
        bar();
        return 1;
        }
    else
        return 0;
    }
```

Formatting styles

Allman (ANSI)

```
int Foo(bool isBar)
{
    if (isBar)
    {
        bar();
        return 1;
    }
    else
        return 0;
}
```

Java

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

K&R

```
int Foo(bool isBar)
{
    if (isBar) {
        bar();
        return 1;
    } else
        return 0;
}
```

Stroustrup

```
int Foo(bool isBar)
{
     if (isBar) {
          bar();
          return 1;
     } else
          return 0;
}
```

Whitesmith

```
int Foo(bool isBar)
    {
    if (isBar)
        {
        bar();
        return 1;
        }
    else
        return 0;
    }
```

Banner

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
        }
    else
        return 0;
    }
```

Formatting styles

Formatting styles

GNU

```
int Foo(bool isBar)
{
  if (isBar)
    {
      bar();
      return 1;
    }
  else
    return 0;
}
```

Formatting styles

GNU

```
int Foo(bool isBar)
{
  if (isBar)
    {
      bar();
      return 1;
    }
  else
    return 0;
}
```

Linux

```
int Foo(bool isBar)
{
        if (isBar) {
                bar();
                return 1;
        } else
                return 0;
}
```

Formatting styles

GNU

```
int Foo(bool isBar)
{
  if (isBar)
    {
      bar();
      return 1;
    }
  else
    return 0;
}
```

Linux

```
int Foo(bool isBar)
{
        if (isBar) {
                bar();
                return 1;
        } else
                return 0;
}
```

Horstmann

```
int Foo(bool isBar)
{  if (isBar)
   {  bar();
      return 1;
   }
   else
      return 0;
}
```

Formatting styles

GNU

```
int Foo(bool isBar)
{
  if (isBar)
    {
      bar();
      return 1;
    }
  else
    return 0;
}
```

Linux

```
int Foo(bool isBar)
{
        if (isBar) {
                bar();
                return 1;
        } else
                return 0;
}
```

Horstmann

```
int Foo(bool isBar)
{   if (isBar)
    {   bar();
        return 1;
    }
    else
        return 0;
}
```

1TBS

```
int Foo(bool isBar)
{
    if (isFoo) {
        bar();
        return 1;
    } else {
        return 0;
    }
}
```

Formatting styles

GNU

```
int Foo(bool isBar)
{
  if (isBar)
    {
      bar();
      return 1;
    }
  else
    return 0;
}
```

Linux

```
int Foo(bool isBar)
{
        if (isBar) {
                bar();
                return 1;
        } else
                return 0;
}
```

Horstmann

```
int Foo(bool isBar)
{  if (isBar)
   {  bar();
      return 1;
   }
   else
      return 0;
}
```

1TBS

```
int Foo(bool isBar)
{
    if (isFoo) {
        bar();
        return 1;
    } else {
        return 0;
    }
}
```

Pico

```
int Foo(bool isBar)
{  if (isBar)
   {  bar();
      return 1; }
    else
      return 0; }
```

Formatting styles

GNU

```
int Foo(bool isBar)
{
  if (isBar)
    {
      bar();
      return 1;
    }
  else
    return 0;
}
```

Linux

```
int Foo(bool isBar)
{
        if (isBar) {
                bar();
                return 1;
        } else
                return 0;
}
```

Horstmann

```
int Foo(bool isBar)
{  if (isBar)
   {  bar();
      return 1;
   }
   else
      return 0;
}
```

1TBS

```
int Foo(bool isBar)
{
    if (isFoo) {
        bar();
        return 1;
    } else {
        return 0;
    }
}
```

Pico

```
int Foo(bool isBar)
{  if (isBar)
   {  bar();
      return 1; }
    else
      return 0; }
```

Lisp

```
int Foo(bool isBar) {
    if (isBar) {
        bar()
        return 1; }
    else
        return 0; }
```

Compact formatting

Compact formatting

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    }
    else {
        return 0;
    }
}
```

Compact formatting

```
int Foo(bool isBar) {
    if (isBar) {
        bar();
        return 1;
    }
    else {
        return 0;
    }
}
```

Tabs take less space and are harder to mishandle.

Questions?

Headers

Headers

Two kinds of header files:

Headers

Two kinds of header files:

<header>	System header files (libraries)
"(path/)header.hpp"	Project header files

Headers

Two kinds of header files:

<header>	System header files (libraries)
"(path/)header.hpp"	Project header files

Dividing declarations and definition.

Headers

Two kinds of header files:

<header>	System header files (libraries)
"(path/)header.hpp"	Project header files

Dividing declarations and definition.
Cleaning up the code.

Headers

Two kinds of header files:

<header>	System header files (libraries)
"(path/)header.hpp"	Project header files

Dividing declarations and definition.
Cleaning up the code.
Reusability.

Headers

Two kinds of header files:

<header>	System header files (libraries)
"(path/)header.hpp"	Project header files

Dividing declarations and definition.
Cleaning up the code.
Reusability.
Every cpp got its own hpp.

Headers

Two kinds of header files:

<header>	System header files (libraries)
"(path/)header.hpp"	Project header files

Dividing declarations and definition.
Cleaning up the code.
Reusability.
Every cpp got its own hpp.
Preprocessor variables come in all uppercase.

Preprocessor directives

Preprocessor directives

Executed before compilation

Preprocessor directives

Executed before compilation

#define Defines a preprocessor variable, induces replaces.

#undef Undefines a preprocessor variable.

Preprocessor directives

Executed before compilation

#define Defines a preprocessor variable, induces replaces.

#undef Undefines a preprocessor variable.

#if Checks condition (known to preprocessor).

#elif Checks another condition.

#else Other cases.

#endif Closes if block.

Preprocessor directives

Executed before compilation

#define Defines a preprocessor variable, induces replaces.

#undef Undefines a preprocessor variable.

#if Checks condition (known to preprocessor).

#elif Checks another condition.

#else Other cases.

#endif Closes if block.

#ifdef Checks existence of definition.

#ifndef Checks inexistence of definition.

Preprocessor directives

Executed before compilation

#define	Defines a preprocessor variable, induces replaces.
#undef	Undefines a preprocessor variable.
#if	Checks condition (known to preprocessor).
#elif	Checks another condition.
#else	Other cases.
#endif	Closes if block.
#ifdef	Checks existence of definition.
#ifndef	Checks inexistence of definition.
#include	Includes another file (usually a header).

Preprocessor directives

Executed before compilation

#define Defines a preprocessor variable, induces replaces.

#undef Undefines a preprocessor variable.

#if Checks condition (known to preprocessor).

#elif Checks another condition.

#else Other cases.

#endif Closes if block.

#ifdef Checks existence of definition.

#ifndef Checks inexistence of definition.

#include Includes another file (usually a header).

#line Changes number of line and name of file (for compiler only).

#error Prints out error and ends compilation.

#pragma Sets various compiler options.

Header shield

Header shield

```
#ifndef HEADER_HPP
#define HEADER_HPP

#include <header1>
#include <header2>
#include "header3.hpp"

 class c {
        ...
 };

 int func1(int);
 double func2(char);

#endif
```

Questions?

I/O streams

I/O streams

In UNIX, everything is a file.

I/O streams

In UNIX, everything is a file.
Devices are files too!

I/O streams

In UNIX, everything is a file.
Devices are files too!
C++ communicates using streams.

I/O streams

In UNIX, everything is a file.
Devices are files too!
C++ communicates using streams.

- Keyboard is standard input stream, cin &0

I/O streams

In UNIX, everything is a file.
Devices are files too!
C++ communicates using streams.

- Keyboard is standard input stream, cin &0
- Terminal is standard output stream, cout &1

I/O streams

In UNIX, everything is a file.
Devices are files too!
C++ communicates using streams.

- Keyboard is standard input stream, cin &0
- Terminal is standard output stream, cout &1
- Terminal is also standard error stream, cerr &2

Standard input stream

Standard input stream

cin is an instance of istream from iostream library.

Standard input stream

cin is an instance of istream from iostream library.

operator» Pushes data from stream to target variable.

Standard input stream

cin is an instance of istream from iostream library.

operator»	Pushes data from stream to target variable.
fail()	Checks for error in stream.

Standard input stream

cin is an instance of istream from iostream library.

operator»	Pushes data from stream to target variable.
fail()	Checks for error in stream.
get()	Gets a character (or more) from stream.

Standard input stream

cin is an instance of istream from iostream library.

operator» Pushes data from stream to target variable.

fail() Checks for error in stream.

get() Gets a character (or more) from stream.

getline() Gets a line from stream.

Standard input stream

cin is an instance of istream from iostream library.

operator» Pushes data from stream to target variable.

fail() Checks for error in stream.

get() Gets a character (or more) from stream.

getline() Gets a line from stream.

eof() Checks if the stream is open.

Standard input stream

cin is an instance of istream from iostream library.

operator»	Pushes data from stream to target variable.
fail()	Checks for error in stream.
get()	Gets a character (or more) from stream.
getline()	Gets a line from stream.
eof()	Checks if the stream is open.
wcin	Wide stream for Unicode.

Standard output stream

Standard output stream

cout is instance of ostream from iostream library.

Standard output stream

cout is instance of ostream from iostream library.

operator« Pushes data to stream.

Standard output stream

cout is instance of ostream from iostream library.

operator« Pushes data to stream.

fail() Checks for error in stream.

Standard output stream

cout is instance of ostream from iostream library.

operator« Pushes data to stream.

fail() Checks for error in stream.

put() Puts a character (or more) to stream.

Standard output stream

cout is instance of ostream from iostream library.

operator« Pushes data to stream.

fail() Checks for error in stream.

put() Puts a character (or more) to stream.

write() Puts a line to stream.

Standard output stream

cout is instance of ostream from iostream library.

operator«	Pushes data to stream.
fail()	Checks for error in stream.
put()	Puts a character (or more) to stream.
write()	Puts a line to stream.
eof()	Checks if the stream is open.

Standard output stream

cout is instance of ostream from iostream library.

operator«	Pushes data to stream.
fail()	Checks for error in stream.
put()	Puts a character (or more) to stream.
write()	Puts a line to stream.
eof()	Checks if the stream is open.
wcout	Wide stream for Unicode.

Manipulators

Manipulators

Most in iomanip library

Manipulators

Most in iomanip library

- endl

Manipulators

Most in iomanip library

- endl
- dec, hex, oct

Manipulators

Most in iomanip library

- endl
- dec, hex, oct
- fixed

Manipulators

Most in iomanip library

- endl
- dec, hex, oct
- fixed
- setw(<int>)
- setfill(<char>)

Manipulators

Most in iomanip library

- endl
- dec, hex, oct
- fixed
- setw(<int>)
- setfill(<char>)

```
std::cout « std::setiosflags(std::ios::uppercase);
std::cout.flags(std::ios::uppercase);
```

Questions?

Arrays

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

<type><identifier>[<size>]. . .

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

<type><identifier>[<size>]. . .

Size is fixed for the whole life of the array

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

<type><identifier>[<size>]. . .

Size is fixed for the whole life of the array

Arrays are indexed from 0, ending in size - 1

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

<type><identifier>[<size>]. . .

Size is fixed for the whole life of the array

Arrays are indexed from 0, ending in size - 1

Multidimensional arrays can be accessed both by several indices and by just one index

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

<type><identifier>[<size>]. . .

Size is fixed for the whole life of the array

Arrays are indexed from 0, ending in size - 1

Multidimensional arrays can be accessed both by several indices and by just one index

More on that later with pointers

Arrays

Arrays are data structures, allowing several instances of one type under one identifier

Any dimensionality, just like tensors http://en.wikipedia.org/wiki/Tensor

Usually represent some kind of linked data

<type><identifier>[<size>]. . .

Size is fixed for the whole life of the array

Arrays are indexed from 0, ending in size - 1

Multidimensional arrays can be accessed both by several indices and by just one index

More on that later with pointers

```
#include <iostream>

int main() {
    int size = 5;
    int a[size];
    for (int i = 0; i < size; i++) {
        a[i] = i;
        std::cout << a[i] << std::endl;
    }
    return 0;
}
```

Visibility of variables

Visibility of variables

Variable is always visible (and exists) only in the block, in which it was created (and all subblocks).

Visibility of variables

Variable is always visible (and exists) only in the block, in which it was created (and all subblocks).
Variables created outside any block are called global, all the other are called local (to given block).

Visibility of variables

Variable is always visible (and exists) only in the block, in which it was created (and all subblocks).
Variables created outside any block are called global, all the other are called local (to given block).
Global variables are considered bad design, especially in OOP (C++) and you shouldn't use them.

Visibility of variables

Variable is always visible (and exists) only in the block, in which it was created (and all subblocks).

Variables created outside any block are called global, all the other are called local (to given block).

Global variables are considered bad design, especially in OOP (C++) and you shouldn't use them.

Variables can overshadow other variables with the same name.

Visibility of variables

Variable is always visible (and exists) only in the block, in which it was created (and all subblocks).
Variables created outside any block are called global, all the other are called local (to given block).
Global variables are considered bad design, especially in OOP (C++) and you shouldn't use them.
Variables can overshadow other variables with the same name.
More local one always wins.

Visibility of variables

Variable is always visible (and exists) only in the block, in which it was created (and all subblocks).
Variables created outside any block are called global, all the other are called local (to given block).
Global variables are considered bad design, especially in OOP (C++) and you shouldn't use them.
Variables can overshadow other variables with the same name.
More local one always wins.

```
#include <iostream>

int main() {
    int a = 2;
    std::cout << a << std::endl;
    if (1) {
        int a = 3;
        std::cout << a << std::endl;
    }
    std::cout << a << std::endl;
    return 0;
}
```

Function overloading

Function overloading

Just as variables can be overshadowed, functions can be overloaded.

Function overloading

Just as variables can be overshadowed, functions can be overloaded.
(Don't mix it up!)

Function overloading

Just as variables can be overshadowed, functions can be overloaded.
(Don't mix it up!)
Overloading doesn't work on base of locality, but rather on difference of input types.

Function overloading

Just as variables can be overshadowed, functions can be overloaded.
(Don't mix it up!)
Overloading doesn't work on base of locality, but rather on difference of input types.

```
int add(int, int);
double add(int, double);
double add(double, int);
double add(double, double);
```

Silicon Hill

Questions?

Break!