Introduction to Computer and Programming

Chapter 9: Introduction to C++

Manuel

Fall 2018

Outline

The birth of C++

Background information:

- Author: Bjarne Stroustrup
- Motivation: other languages are either too low level or too slow

Timeline:

- 1979: C with classes
- 1983: name changed for C++
- 1985: first commercial implementation of C++
- 1989: updated version, C++2.0
- 2011: new version, C++11, enlarged standard library
- 2014: C++14, bug fixes, minor improvements

C++ in a few words

Simple description:

- Compiled programming language
- General-purpose programming language
- Intermediate level language
- Object-oriented programming language

C++ in a few words

Simple description:

- Compiled programming language
- General-purpose programming language
- Intermediate level language
- Object-oriented programming language

Highlights:

- Higher level than C, but still performant
- Code often shorter and cleaner than in C
- Safer: more errors caught at compile time
- No runtime overhead

Outline

C vs. C++

What C++ brings:

- Almost all the aspects of C are preserved
- New features are added
- Sophisticated programs are easier to code
- C++ is almost a superset of C

C vs. C++

What C++ brings:

- Almost all the aspects of C are preserved
- New features are added
- Sophisticated programs are easier to code
- C++ is almost a superset of C

Is this program written in C or C++?

prg.cpp

```
1 #include <stdio.h>
2
3 int main () {
4   int a=5;
5   printf("%d\n",a);
6 }
```

Why easier?

A new approach:

- Easier to manage memory
- New features for generic programming
- Object oriented programming:
  - Variables are defined in term of objects
  - Objects are close from human thinking
  - An object is similar to a structure in C with more “abilities”

Why easier?

A new approach:

- Easier to manage memory
- New features for generic programming
- Object oriented programming:
  - Variables are defined in term of objects
  - Objects are close from human thinking
  - An object is similar to a structure in C with more “abilities”

Programmers can focus more on the problem rather than on how to explain it to the computer

Basics

C++ syntax is similar to C's:

- Function declaration
- Blocks
- For loop
- While loop
- If statement
- Switch statement
- Shorthand operators
- Logical operators
- Short-circuit operators
- Conditional ternary operator

Basics

C++ syntax is similar to C's:

- Function declaration
- Blocks
- For loop
- While loop
- If statement
- Switch statement
- Shorthand operators
- Logical operators
- Short-circuit operators
- Conditional ternary operator

A small difference between C and C++:

C

Implicit assignment from *void

```c
int *x = \
malloc(sizeof(int)*10);
```

C++

No implicit assignment from *void

```c
int *x = \
(int *)  malloc(sizeof(int)*10);
```

Outline

New data type and headers

New in C++:

- New datatype:

```
1 bool a=true, b=false;
```

- New headers:

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

New data type and headers

New in C++:

- New datatype:

```
1 bool a=true, b=false;
```

- New headers:

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

Namespace:

- C: function names conflicts among different libraries
- C++: introduction of *namespace*
- Each library or program has its own namespace
- Namespace for the standard library: `std`

New input/output style

Handling I/O without `printf` and `scanf`:

- Input: `cin >> x`
- Output: `cout << "String"`

New input/output style

Handling I/O without `printf` and `scanf`:

- Input: `cin >> x`
- Output: `cout << "String"`

Example.

input-pb.cpp

```cpp
#include <iostream>
using namespace std;
void TestInput(){
  int x = 0;
  do {
    cout << "Enter a number (-1 to quit): "; cin >> x;
    if(x != -1) cout << x << " was entered" << endl;
  }  while(x != -1);
  cout << "Exit" << endl;
}
int main() {TestInput(); return 0;}
```

Input

Problem with the previous code: input a letter...and exit

input-ok1.cpp

```cpp
#include <iostream>
using namespace std;
void TestInput(){
  int x = 0;
  do {
    cout << "Enter a number (-1 to quit): ";
    if(!(cin >> x)) {
      cout << "The input stream broke!" << endl;
      x = -1;
    }
    if(x != -1) cout << x << " was entered" << endl;
  } while(x != -1);
  cout << "Exit" << endl;
}
int main() {TestInput(); return 0;}
```

Input

Problem with the previous code: the program exits "unexpectedly"

input-ok2.cpp

```cpp
#include <iostream>
using namespace std;
void TestInput(){
  int x=0;
  do {
    cout << "Enter a number (-1 to quit): ";
    cin >> x;
    cin.clear();
    cin.ignore(10000,'\n');
    if(x != -1) cout << x << " was entered" << endl;
  }  while(x != -1);
  cout << "Exit" << endl;
}
int main() {TestInput(); return 0;}
```

Formatting output

Nicer display:

- Width: `setw(width)`
- Prefix: `setfill('z')`
- Alignment: `setiosflags(ios::left)`
- Precision: `setprecision(2)`

Formatting output

Nicer display:

- Width: `setw(width)`
- Prefix: `setfill('z')`
- Alignment: `setiosflags(ios::left)`
- Precision: `setprecision(2)`

Example.

date.cpp

```cpp
#include <iostream>
#include <iomanip>
using namespace std;
void showDate(int m, int d, int y) {
  cout.fill('0');
  cout << setw(2) << m << '/' << setw(2) << d << '/' << setw(4) << y << endl;
}
int main(){
  showDate(6,19,2014);
  cout << setprecision(3) << 1.2249 << endl;
  cout << setprecision(3) << 1.22549 << endl;
}
```

Operator and function overloading

Note on the operators:

- What are << and >> in C?
- What about `cin >> x` or `cout << x`?
- An operator can be reused with a different meaning

Operator and function overloading

Note on the operators:

- What are << and >> in C?
- What about `cin >> x` or `cout << x`?
- An operator can be reused with a different meaning

Similar concept: function overloading

fo.cpp

```
1 #include <iostream>
2 using namespace std;
3 double f(double a);
4 int f(int a);
5 int main () {cout << f(2) << endl; cout << f(2.3) << endl;}
6 double f(double a) {return a;}
7 int f(int a) {return a;}
```

Pointers

No more `malloc, calloc` and `free`:

- Memory for a variable: `int *p = new int;`
- Memory for an array: `int *p = new int[10];`
- Array size can be a variable (not recommended in C)
- Return `NULL` on failure
- Release the memory: `delete p` or `delete[] p`

Pointers

No more `malloc`, `calloc` and `free`:

- Memory for a variable: `int *p = new int;`
- Memory for an array: `int *p = new int[10];`
- Array size can be a variable (not recommended in C)
- Return `NULL` on failure
- Release the memory: `delete p` or `delete[] p`

Any allocated memory must be released

Strings

Improvements on strings:

- Strings in C: array of characters
- Many limitations, low level manipulations
- New type in C++: string

Strings

Improvements on strings:

- Strings in C: array of characters
- Many limitations, low level manipulations
- New type in C++: string

```
1 #include <string>
2 string g="good "; string m="morning";
3 cout << g + m + "!\n";
```

Strings

Improvements on strings:

- Strings in C: array of characters
- Many limitations, low level manipulations
- New type in C++: string

```
1 #include <string>
2 string g="good "; string m="morning";
3 cout << g + m + "!\n";
```

More possibilities: search and learn how to use strings in C++

File I/O

Requires header: `#include <fstream>`

- Open file for reading: `ifstream in("file.txt")`
- Read from a file: `in` used in the same way as `cin`
- Open a file for writing: `ofstream out("file.txt")`
- Write in a file: `out` used in the same way as `cout`
- Read from a file, line by line: `getline(in,s)`

File I/O

Example.
Copy the content of a text file into another text file and display each line on the console output

File I/O

Example.
Copy the content of a text file into another text file and display each line on the console output

fio.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5  void FileIO() {
6    string s;
7    ifstream a("1.txt"); ofstream b("2.txt");
8    while(getline(a,s)) {b << s << endl;  cout << s;}
9  }
10 int main () {FileIO();return 0;}
```

File I/O

What was wrong with the previous code?

fio-c.cpp

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5  void FileIO(){
6    string s;
7    ifstream a("1.txt"); ofstream b("2.txt",ios::app);
8    if (a.is_open() && b.is_open()) {
9      while(getline(a,s)) {b << s << endl; cout << s;}
10     b.close(); a.close();
11   }
12   else cerr << "Unable to open the file(s)\n";
13 }
14 int main () {FileIO();return 0;}
```

Defining constants

C

- `#define PI 3.14`
- Handled early in compilation
- No record of PI at compile time

C++

- `static const float PI=3.14;`
- `PI` is a constant, value cannot be changed
- `PI` is known by the compiler, present in the symbol table
- Type safe

Inline functions

C

- Macros
- Macros expanded early in the compilation
- Hard to debug
- Sides effect with complex macros

C++

- Inline functions
- Treated by the compiler
- Similar as a regular function
- Does not call the function but write a copy of it instead
- Increase size of the program

```
1 inline int sq(int x) { return x*x; }
```

Key points

- What is the difference between C and C++?
- Cite a few novelties
- How to handle input/output?
- How to handle pointers?
- What are operator and function overloading?

Thank you!