6.S096 Lecture 4 – Style and Structure

Transition from C to C++

Andre Kessler

Outline

Assignment Recap

Floating point

- Layout of 32-bit floating point numbers in memory.

Matrix multiply (1 & 2)

- Static vs. dynamic allocation
- Using a data structure
- Optimizing for the cache

Transposition cipher

- Safely allocating and reading an arbitrary-size string
- Test data

Header files

Separation of declaration and definition

```
// in header (.h)
void increment( int *a );
// in source (.c)
void increment( int *a ) {
  ++*a;
}
```

Include guards (file myheader.h)

```
#ifndef _MYHEADER_H
#define _MYHEADER_H
void increment( int *a );
// ...your header stuff
#endif // _MY_HEADER_H
```

Coding style

Consistency and clarity above all.

Vertical space is precious; horizontal space is good

Always using { and } even if not needed

Again: be consistent.

The Minimal C Program

`nothing.c`: takes no arguments, does nothing, returns 0 ("exit success")

```
int main(void) {
  return 0;
}
```

1. To compile: `make nothing`
2. Previous step produced an executable named `nothing`
3. To run: `./nothing`

The Minimal C++ Program

`nothing.cpp`: takes no arguments, does nothing, returns 0.

```
int main() {
  return 0;
}
```

1. To compile: `make nothing`
2. Previous step produced an executable named `nothing`
3. To run: `./nothing`

Hello, world!

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
int main() {
  return 0;
}
```

Hello, world!

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <cstdio>

int main() {
  return 0;
}
```

Hello, world!

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <cstdio>

int main() {
  printf( "Hello, world!\n" );
  return 0;
}
```

Hello, world!

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <cstdio>

int main() {
  printf( "Hello, world!\n" );
  return 0;
}
```

1. To compile: `make hello`
2. Previous step produced an executable named `hello`
3. To run: `./hello`

Hello, world!

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <cstdio>

int main() {
  printf( "Hello, world!\n" );
  return 0;
}
```

1. To compile: `make hello`
2. Previous step produced an executable named `hello`
3. To run: `./hello`
4. `Hello, world!`

Hello, world!

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <cstdio>

int main() {
  printf( "Hello, world!\n" );
  return 0;
}
```

1. To compile: `make hello`
2. Previous step produced an executable named `hello`
3. To run: `./hello`
4. `Hello, world!`

Hello, world! done better

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <iostream>

int main() {
  std::cout << "Hello, world!\n";
  return 0;
}
```

1. To compile: `make hello`
2. Previous step produced an executable named `hello`
3. To run: `./hello`

Hello, world! done better

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <iostream>

int main() {
  std::cout << "Hello, world!\n";
  return 0;
}
```

1. To compile: `make hello`
2. Previous step produced an executable named `hello`
3. To run: `./hello`
4. `Hello, world!`

Hello, world! done better

`hello.cpp`: takes no arguments, prints "Hello, world!", returns 0

```
#include <iostream>

int main() {
  std::cout << "Hello, world!\n";
  return 0;
}
```

1. To compile: `make hello`
2. Previous step produced an executable named `hello`
3. To run: `./hello`
4. `Hello, world!`

What is C++?

- **Multiparadigm programming language**
- Has procedural, object-oriented, functional, generic, and metaprogramming features.
- Procedural: underneath the surface, C++ is mostly C
- Object-oriented: classes, encapsulation, inheritance, and polymorphism
- Generic: template metapgramming, programs that run during compilation
- Standard Template Library (STL): common idioms, containers, and algorithms

The Key Differences

Generally, can compile C code with a C++ compiler

References vs pointers

C++ is more strongly typed

- Stronger notions of casting, `static_cast<>` and so on
- At the same time, `auto` variables

C++ supports notions of immutability

- `const` correctness and `constexpr`.

C++ has extensive abstraction mechanisms

- Classes and wrapping our memory management
- Templates `template<typename T>`

Immutability and References

In addition to the good old pointer, C++ also has a notion of a reference. Let's look at an increment function written in C:

```
void increment( int *a ) {
  ++*a;
}
// later: call it with
int a = 5;
increment( &a );
```

The C++ way:

```
void increment( int &a ) {
  ++a;
}
// later: call with
int a = 5;
increment( a );
```

An example of abstraction

A safer array

```
// Interface
class Array {
  size_t _size;
  double *_elem;
public:
  Array( size_t theSize );
  ~Array();

  inline size_t size() const { return _size; };
  double& operator[]( size_t i );
};
```

C++ Resources

The C++ Programming Language, 4th ed. by Bjarne Stroustrop
Effective C++, *More Effective C++*, and *Effective STL* by Scott Meyers
http://cplusplus.com
http://cppreference.com

Examples

Time for some examples...

Second Assignment

Second assignment is posted before midnight: three problems total 1000 points

- linklist (300, C)
- geom (300, C++)
- mst (400, C++)

Wrap-up & Monday

Class on Friday

- Object-oriented programming in C++

Questions?

6.S096 Effective Programming in C and C++

IAP 2014