CSCI 2270: Data Structures

Lecture 02: C++ Review

Ashutosh Trivedi
ashutosh.trivedi@colorado.edu

Department of Computer Science
University of Colorado Boulder

C++: A quick review

Recommended C++ Resources

1. Bjarne Stroustrup, *The C++ Programming Language*, 4-th Edition.
   /* C++ Reference by the inventor of the lanaguage */
2. Scott Meyers, *Effective C++*
   /* C++ specific tips to improve programs and designs */
3. Scott Meyers, *Effective Modern C++*
   /* Similar tips extended to C++11 and C++14 versions */
4. Misfeldt, Bumgardner, and Gray, *The Elements of C++ Style*
   /* "Strunk and White" of writing human-readable C++ code */
5. Online Resources:
   - `http://www.cplusplus.com/doc/tutorial/`
   - `https://en.cppreference.com/`
   - `https://www.geeksforgeeks.org/c-plus-plus/`

Why C++?

- Popular and relevant (from last 20 years):
  - End-user applications (Word, Excel, Powerpoint, Photoshop, Acrobat, Doom 3, Web-browsers, and so on)

Why C++?

- Popular and relevant (from last 20 years):
  - End-user applications (Word, Excel, Powerpoint, Photoshop, Acrobat, Doom 3, Web-browsers, and so on)
  - Operating systems (Windows, OS X, Linux – some versions of C/C++)

Why C++?

- Popular and relevant (from last 20 years):
  - End-user applications (Word, Excel, Powerpoint, Photoshop, Acrobat, Doom 3, Web-browsers, and so on)
  - Operating systems (Windows, OS X, Linux – some versions of C/C++)
  - Database software and large-scale web-applications (MySQL, Amazon, Google, Wikipedia, etc.)
  - Device drivers, numerical computations, and many more...

Why C++?

- Popular and relevant (from last 20 years):
  - End-user applications (Word, Excel, Powerpoint, Photoshop, Acrobat, Doom 3, Web-browsers, and so on)
  - Operating systems (Windows, OS X, Linux – some versions of C/C++)
  - Database software and large-scale web-applications (MySQL, Amazon, Google, Wikipedia, etc.)
  - Device drivers, numerical computations, and many more...
- Stable, compatible, and scalable.

C Vs C++?

- C++ is C incremented. /* C with classes */
- C++ is backward-compatible with C. /* some minor exceptions! */
- C++ is more expressive. /* fewer lines of code required! */
- C++ is just as permissive. /* can do anything that C can! */
- C++ is just as efficient. /* lets you manipulate bits directly! */
- C++ is more maintainable. /* Due to structure and elegance enabled by object-oriented features! */

Design Philosophy: by Bjarne Stroustrup

Programming languages typically serve two purposes:

1. *as a vehicle for specifying actions to be executed.*

/* close to machine. */

Design Philosophy: by Bjarne Stroustrup

Programming languages typically serve two purposes:

1. *as a vehicle for specifying actions to be executed.*

   /* close to machine. */

2. *as set of concepts to help designer think about what can be done.*

   /* close to the problem being solved. */

Design Philosophy: by Bjarne Stroustrup

Programming languages typically serve two purposes:

1. *as a vehicle for specifying actions to be executed.*

   /* close to machine. */

2. *as set of concepts to help designer think about what can be done.*

   /* close to the problem being solved. */

Object-oriented C++ excels at both.

C++ Programming paradigms

1. procedural: *implements algorithms via functions.*

C++ Programming paradigms

1. procedural: *implements algorithms via functions.*
2. modular: *partition programs into modules (separate compilation)*

C++ Programming paradigms

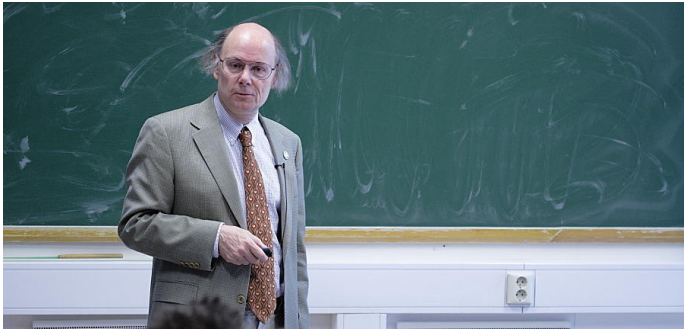
1. procedural: *implements algorithms via functions.*
2. modular: *partition programs into modules (separate compilation)*
3. object-oriented: *divide problem into natural data-structures (classes with data-hiding and inheritance)*

C++ Programming paradigms

1. procedural: *implements algorithms via functions.*
2. modular: *partition programs into modules (separate compilation)*
3. object-oriented: *divide problem into natural data-structures (classes with data-hiding and inheritance)*
4. abstract: *separate interface form implementation (abstract classes)*

C++ Programming paradigms

1. procedural: *implements algorithms via functions.*
2. modular: *partition programs into modules (separate compilation)*
3. object-oriented: *divide problem into natural data-structures (classes with data-hiding and inheritance)*
4. abstract: *separate interface form implementation (abstract classes)*
5. generic: *generic algorithms to manipulate arbitrary data-type (STL: containers, algorithms)*



" Don't panic." Bjarne Stroustrup, Creator of C++.

" The only way to learn a new programming language is by writing programs in it."

Dennis Ritchie (1941-2011), Creator of C.

“Hello, World!” in C++ as a C program

```cpp
// program1.cpp
#include<stdio.h>
int main()
{
  printf("Hello, World!\n");
  return 100;
}
```

“Hello, World!” in C++ as a C program

```
// program1.cpp
#include<stdio.h>
int main()
{
  printf("Hello, World!\n");
  return 100;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program1.cpp -o hello1
./hello1
```

“Hello, World!” in C++

```cpp
// program2.cpp
#include<iostream>
int main()
{
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

“Hello, World!” in C++

```cpp
// program2.cpp
#include<iostream>
int main()
{
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program2.cpp -o hello2
./hello2
```

“Hello, World!” in C++

```cpp
// program2.cpp
#include<iostream>
int main()
{
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program2.cpp -o hello2
./hello2
```

1. Like the cstdio header inherited from C’s stdio.h, iostream provides basic input and output services for C++ programs.

"Hello, World!" in C++

```cpp
// program2.cpp
#include<iostream>
int main()
{
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program2.cpp -o hello2
./hello2
```

1. Like the cstdio header inherited from C's stdio.h, iostream provides basic input and output services for C++ programs.
2. Namespaces allow one to reuse names across different libraries. The namespace `std` refers to standard namespace. You can obviate the need for using `std::` with standard streams cin and cout by declaring `using namespace std` in your program.

Command line arguments in C++

```cpp
// program3.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  std::cout << "Hello,";
  for (int i=1; i < argc; i++) std::cout << " " << argv[i] ;
  std::cout << "!" << std::endl;
  return 0;
}
```

Command line arguments in C++

```cpp
// program3.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  std::cout << "Hello,";
  for (int i=1; i < argc; i++) std::cout << " " << argv[i] ;
  std::cout << "!" << std::endl;
  return 0;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program3.cpp -o hello3
./hello3
./hello3 Ashutosh Asa Maciej
```

Command line arguments in C++

```cpp
// program3.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  std::cout << "Hello,";
  for (int i=1; i < argc; i++) std::cout << " " << argv[i] ;
  std::cout << "!" << std::endl;
  return 0;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program3.cpp -o hello3
./hello3
./hello3 Ashutosh Asa Maciej
```

1. `argc` (argument count): the number of argument to this program and `argv[]` (argument vector): the array of character pointers (strings) containing arguments.

Command line arguments in C++

```cpp
// program3.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  std::cout << "Hello,";
  for (int i=1; i < argc; i++) std::cout << " " << argv[i] ;
  std::cout << "!" << std::endl;
  return 0;
}
```

```
#!/usr/local/bin/bash
# Shell script to compile and execute
g++ program3.cpp -o hello3
./hello3
./hello3 Ashutosh Asa Maciej
```

1. `argc` (argument count): the number of argument to this program and `argv[]` (argument vector): the array of character pointers (strings) containing arguments.
2. What is unpleasant about this program, and how do you fix it?

Fundamental Types and their sizes

```cpp
// program4.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  bool bo = true;
  unsigned char ch = 'a'; // signed and unsigned
  long int in = 100; // signed and unsigned, short and long
  long double fl = 1.2e10; // float, double, and long double
  std::cout << "Size of:" << std::endl;
  std::cout << "\t bool(" << sizeof(bo) << ")" << std::endl;
  std::cout << "\t char(" << sizeof(ch) <<")" << std::endl;
  std::cout << "\t int(" << sizeof(in) << ")" << std::endl;
  std::cout << "\t float(" << sizeof(fl) << ")\n";
  return 0;
}
```

Fundamental Types and their sizes

```cpp
// program4.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  bool bo = true;
  unsigned char ch = 'a'; // signed and unsigned
  long int in = 100; // signed and unsigned, short and long
  long double fl = 1.2e10; // float, double, and long double
  std::cout << "Size of:" << std::endl;
  std::cout << "\t bool(" << sizeof(bo) << ")" << std::endl;
  std::cout << "\t char(" << sizeof(ch) <<")" << std::endl;
  std::cout << "\t int(" << sizeof(in) << ")" << std::endl;
  std::cout << "\t float(" << sizeof(fl) << ")\n";
  return 0;
}
```

```
g++ program4.cpp -o sizes
./sizes
```

Enumeration Type and its size

```cpp
// program5.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  // Enums: user-defined types
  enum exams {MIDTERM1, MIDTERM2, FINAL, PROJECT};
  exams ex = MIDTERM1;
  std::cout << "Size of exams (" << sizeof(ex);
  std::cout << ")" << std::endl;
  std::cout << "Size of exams (" << sizeof(exams);
  std::cout << ")" << std::endl;
  return 0;
}
```

Enumeration Type and its size

```
// program5.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  // Enums: user-defined types
  enum exams {MIDTERM1, MIDTERM2, FINAL, PROJECT};
  exams ex = MIDTERM1;
  std::cout << "Size of exams (" << sizeof(ex);
  std::cout << ")" << std::endl;
  std::cout << "Size of exams (" << sizeof(exams);
  std::cout << ")" << std::endl;
  return 0;
}
```

1. An *enumeration* is a use-defined type that can hold a set of values specified by the user.

Enumeration Type and its size

```cpp
// program5.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  // Enums: user-defined types
  enum exams {MIDTERM1, MIDTERM2, FINAL, PROJECT};
  exams ex = MIDTERM1;
  std::cout << "Size of exams (" << sizeof(ex);
  std::cout << ")" << std::endl;
  std::cout << "Size of exams (" << sizeof(exams);
  std::cout << ")" << std::endl;
  return 0;
}
```

1. An *enumeration* is a use-defined type that can hold a set of values specified by the user.
2. Once defined, it works like an integer type.

Enumeration Types and Switch Statement

```cpp
// program6.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  enum exams {MIDTERM1, MIDTERM2, FINAL, PROJECT};
  exams ex1 = (exams) atoi(argv[1]);
  switch (ex1) {
  case MIDTERM1:
  case MIDTERM2:
    std::cout << "Can improve the grades with finals!";
    break;
  case FINAL:
  case PROJECT:
    std::cout << "Sorry! You cannot improve!";
    break;
  }
  return 0;
}
```

Pointer Variables

```cpp
// program7.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  char ch= 'a';
  char *cp; // cp is a pointer variable
  cp = &ch; // cp points to the address of the ch
  std::cout << "Size of a pointer to char: ";
  std::cout << sizeof(char *) << std::endl;
  std::cout << "Address of ch is = " << (void *) cp;
  return 0;
}
```

1. What are the sizes of pointers to different types of objects?

Pointer Variables

```cpp
// program7.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  char ch= 'a';
  char *cp; // cp is a pointer variable
  cp = &ch; // cp points to the address of the ch
  std::cout << "Size of a pointer to char: ";
  std::cout << sizeof(char *) << std::endl;
  std::cout << "Address of ch is = " << (void *) cp;
  return 0;
}
```

1. What are the sizes of pointers to different types of objects?
2. Repeat the above exercise for other types.

Pointer Variables

```cpp
// program7.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  char ch= 'a';
  char *cp; // cp is a pointer variable
  cp = &ch; // cp points to the address of the ch
  std::cout << "Size of a pointer to char: ";
  std::cout << sizeof(char *) << std::endl;
  std::cout << "Address of ch is = " << (void *) cp;
  return 0;
}
```

1. What are the sizes of pointers to different types of objects?
2. Repeat the above exercise for other types.
3. A pointer to variable of type T is:
   - 3.1 `T* p`
   - 3.2 `T *p`

Pointer Variables

```cpp
// program7.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  char ch= 'a';
  char *cp; // cp is a pointer variable
  cp = &ch; // cp points to the address of the ch
  std::cout << "Size of a pointer to char: ";
  std::cout << sizeof(char *) << std::endl;
  std::cout << "Address of ch is = " << (void *) cp;
  return 0;
}
```

1. What are the sizes of pointers to different types of objects?
2. Repeat the above exercise for other types.
3. A pointer to variable of type T is:
   - 3.1 `T* p`
   - 3.2 `T *p`
4. A pointer variable equal to 0 means it does not refer to an object. Use of **NULL** discouraged!

Arrays (Statically Declared Arrays)

```cpp
// program8.cpp
#include<iostream>
int main(int argc, char* argv[])
{
  int ia[3]; //An array of three integers with garbage values
  std::cout << ia[1] << std::endl;
  float fa[] = {1, 2, 3}; //An array of three floats initialzed: size
      automatically computed
  std::cout << fa[1] << std::endl;
  return 0;
}
```

1. Static Array storage is contiguous.
2. Array bound must be a constant expression. If you need variable bounds, use a `vector`.
3. What happens when initialization and array size mismatch?
4. Multi-dimensional arrays (contiguous in row-order fashion!).

Arrays (Dynamically Declared Arrays)

```
1  // program9.cpp
2  #include<iostream>
3  int main(int argc, char* argv[])
4  {
5    int* pa = 0; // pa is a pointer to integers
6    int n;
7    std::cout <<  "Enter dynamically allocated array size:";
8    std::cin >> n;
9    pa = new int[n];
10   for (int i = 0; i < n; i++) {
11     pa[i] = i;
12   }
13   // Use a as a normal array
14   delete[] pa; // When done, free memory pointed to by a.
15   pa = 0; //// Clear a to prevent using invalid memory reference.
16   return 0;
17 }
```

References (A Rose by another name!)

```
1  // program10.cpp
2  #include<iostream>
3  int main(int argc, char* argv[])
4  {
5    int i = 1;
6    int &r = i; // r and i refer to same int
7    // int &s;  // Error: must be initialized unless "extern"
8    int x = r;  // x = 1
9    r = 2;      // x = 2
10   return 0;
11 }
```

Structures (Our first data-structure!)

```cpp
// program11.cpp
#include<iostream>
int main(int argc, char* argv[])  {
  struct address {
    std::string name;
    long int number;
    std::string street;
    std::string town;
    std::string state;
    int zip;
  };
  address myadress = {"Ashutosh Trivedi", 4141, "Spruce Street", "
      Philadelphia", "PA", 19104};
  std::cout << myadress.name << " lives in " << myadress.town << std::endl;
  return 0;
}
```