CSE 333
Lecture 10 - references, const, classes

Today’s goals

Useful C++ features
- references, const

Introducing C++ classes
- defining, using them
Reminder: pointers

C: a pointer is a variable containing an address
- you can change its value to change what it is pointing to
- a pointer can contain the address of some other variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int *z = &x;
    *z += 1; // sets x to 6
    x += 1; // sets x (and therefore *z) to 7
    z = &y; // sets z to the address of y
    *z += 1; // sets y (and therefore *z) to 11
    return EXIT_SUCCESS;
}
```

```
x  5
y  10
z 0xbff2d4
```

Reminder: pointers

C: a pointer is a variable containing an address
- you can change its value to change what it is pointing to
- a pointer can contain the address of a different variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int *z = &x;
    *z += 1; // sets x to 6
    x += 1; // sets x (and therefore *z) to 7
    z = &y; // sets z to the address of y
    *z += 1; // sets y (and therefore *z) to 11
    return EXIT_SUCCESS;
}
```

```
x  5
y  10
z 0xbff2d4
```
Reminder: pointers

C: a pointer is a variable containing an address
- you can change its value to change what it is pointing to
- a pointer can contain the address of a different variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int *z = &x;
    *z += 1; // sets x to 6
    x += 1; // sets x (and therefore *z) to 7
    z = &y; // sets z to the address of y
    *z += 1; // sets y (and therefore *z) to 11
    return EXIT_SUCCESS;
}
```

x 6
y 10
z 0xbfff2d4

Reminder: pointers

C: a pointer is a variable containing an address
- you can change its value to change what it is pointing to
- a pointer can contain the address of a different variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int *z = &x;
    *z += 1; // sets x to 6
    x += 1; // sets x (and therefore *z) to 7
    z = &y; // sets z to the address of y
    *z += 1; // sets y (and therefore *z) to 11
    return EXIT_SUCCESS;
}
```

x 7
y 10
z 0xbfff2d4

pointer.cc
Reminder: pointers

C: a pointer is a variable containing an address
- you can change its value to change what it is pointing to
- a pointer can contain the address of a different variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int *z = &x;
    *z += 1; // sets *z (and therefore x) to 6
    x += 1; // sets x (and therefore *z) to 7
    z = &y; // sets z to the address of y
    *z += 1; // sets *z (and therefore y) to 11
    return EXIT_SUCCESS;
}
```

-x 7
-y 10
-z 0xbfff2d0

Reminder: pointers

C: a pointer is a variable containing an address
- you can change its value to change what it is pointing to
- a pointer can contain the address of a different variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int *z = &x;
    *z += 1; // sets *z (and therefore x) to 6
    x += 1; // sets x (and therefore *z) to 7
    z = &y; // sets z to the address of y
    *z += 1; // sets *z (and therefore y) to 11
    return EXIT_SUCCESS;
}
```

-x 7
-y 11
-z 0xbfff2d0
References

C++: introduces references as part of the language

- a reference acts like an alias for some other variable
  - alias: another name that is bound to the aliased variable
  - mutating a reference is mutating the referenced variable

```cpp
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int &z = x; // binds the name "z" to variable x
    z += 1; // sets z (and thus x) to 6
    x += 1; // sets x (and thus z) to 7
    z = y; // sets z (and thus x) to the value of y
    z += 1; // sets z (and thus x) to 11
    return EXIT_SUCCESS;
}
```

```
x  5
y 10
```

References

C++: introduces references as part of the language

- a reference is an alias for some other variable
  - alias: another name that is bound to the aliased variable
  - mutating a reference is mutating the referenced variable

```cpp
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int &z = x; // binds the name "z" to variable x
    z += 1; // sets z (and thus x) to 6
    x += 1; // sets x (and thus z) to 7
    z = y; // sets z (and thus x) to the value of y
    z += 1; // sets z (and thus x) to 11
    return EXIT_SUCCESS;
}
```

```
x,z  5
y 10
```
References

C++: introduces references as part of the language

- a reference is an alias for some other variable
  - alias: another name that is bound to the aliased variable
  - mutating a reference is mutating the referenced variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int &z = x; // binds the name "z" to variable x
    z += 1; // sets z (and thus x) to 6
    x += 1; // sets x (and thus z) to 7
    z = y;  // sets z (and thus x) to the value of y
    z += 1; // sets z (and thus x) to 11
    return EXIT_SUCCESS;
}
```

x, z  6
y  10

---

References

C++: introduces references as part of the language

- a reference is an alias for some other variable
  - alias: another name that is bound to the aliased variable
  - mutating a reference is mutating the referenced variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int &z = x; // binds the name "z" to variable x
    z += 1; // sets z (and thus x) to 6
    x += 1; // sets x (and thus z) to 7
    z = y;  // sets z (and thus x) to the value of y
    z += 1; // sets z (and thus x) to 11
    return EXIT_SUCCESS;
}
```

x, z  7
y  10
References

C++: introduces references as part of the language

- a reference is an alias for some other variable
  - alias: another name that is bound to the aliased variable
  - mutating a reference is mutating the referenced variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int &z = x; // binds the name "z" to variable x
    z += 1; // sets z (and thus x) to 6
    x += 1; // sets x (and thus z) to 7
    z = y;  // sets z (and thus x) to the value of y
    z += 1; // sets z (and thus x) to 11
    return EXIT_SUCCESS;
}
```

```c
x, z 10
```

```c
y 10
```

References

C++: introduces references as part of the language

- a reference is an alias for some other variable
  - alias: another name that is bound to the aliased variable
  - mutating a reference is mutating the referenced variable

```c
int main(int argc, char **argv) {
    int x = 5, y = 10;
    int &z = x; // binds the name "z" to variable x
    z += 1; // sets z (and thus x) to 6
    x += 1; // sets x (and thus z) to 7
    z = y;  // sets z (and thus x) to the value of y
    z += 1; // sets z (and thus x) to 11
    return EXIT_SUCCESS;
}
```

```c
x, z 11
```

```c
y 10
```
Pass by reference

C++ allows you to truly pass-by-reference

- client passes in an argument with normal syntax
  - function uses reference parameters with normal syntax
  - modifying a reference parameter modifies the caller’s argument

```cpp
void swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main(int argc, char **argv) {
    int a = 5, b = 10;
    swap(a, b);
    cout << "a: " << a << '; ' << 'b: " << b << endl;
    return EXIT_SUCCESS;
}
```

passbyreference.cc
Pass by reference

C++ allows you to truly pass-by-reference
- client passes in an argument with normal syntax
  - function uses reference parameters with normal syntax
  - modifying a reference parameter modifies the caller's argument

```cpp
void swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main(int argc, char **argv) {
    int a = 5, b = 10;
    swap(a, b);
    cout << "a: " << a << '; ' << b << endl;
    return EXIT_SUCCESS;
}
```

Pass by reference

C++ allows you to truly pass-by-reference
- client passes in an argument with normal syntax
  - function uses reference parameters with normal syntax
  - modifying a reference parameter modifies the caller's argument

```cpp
void swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main(int argc, char **argv) {
    int a = 5, b = 10;
    swap(a, b);
    cout << "a: " << a << '; ' << b << endl;
    return EXIT_SUCCESS;
}
```
Pass by reference

C++ allows you to truly pass-by-reference

- client passes in an argument with normal syntax
  - function uses reference parameters with normal syntax
  - modifying a reference parameter modifies the caller’s argument

```cpp
void swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main(int argc, char **argv) {
    int a = 5, b = 10;

    swap(a, b);
    cout << "a: " << a << " b: " << b << endl;
    return EXIT_SUCCESS;
}
```

```cpp
(main) a 10
(main) b 10
```
Pass by reference

C++ allows you to truly pass-by-reference

- client passes in an argument with normal syntax
  - function uses reference parameters with normal syntax
  - modifying a reference parameter modifies the caller’s argument

```c++
void swap(int &x, int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}

int main(int argc, char **argv) {
    int a = 5, b = 10;
    swap(a, b);
    cout << "a: " << a << "; b: " << b << endl;
    return EXIT_SUCCESS;
}
```

const

const: cannot be changed

- used much more in C++ than in C

```c++
void BrokenPrintSquare(const int &i) {
    i = i*i; // Compiler error here!
    std::cout << i << std::endl;
}

int main(int argc, char **argv) {
    int j = 2;
    BrokenPrintSquare(j);
    return EXIT_SUCCESS;
}
```
**const**

**const’s syntax is confusing**

```c
int main(int argc, char **argv) {
    int x = 5;  // x is an int
    const int y = 6;  // y is a (const int)
y++;  // compiler error

    const int *z = &y;  // z is a (variable pointer) to a (const int)
    *z += 1;  // compiler error
    z++;  // ok

    int *const w = &x;  // w is a (const pointer) to a (variable int)
    *w += 1;  // ok
    w++;  // compiler error

    const int *const v = &x;  // v is a (const pointer) to a (const int)
    *v += 1;  // compiler error
    v++;  // compiler error

    return EXIT_SUCCESS;
}
```

**style guide tip**

- use const reference parameters to pass input
- use pointers to pass output parameters
  - input parameters first, then output parameters last

```c
#include <cstdlib>

void CalcArea(const int &width, const int &height,
    int *const area) {
    *area = width * height;
}

int main(int argc, char **argv) {
    int w = 10, h = 20, a;
    CalcArea(w, h, &a);
    return EXIT_SUCCESS;
}
```
Classes

class declaration syntax (in a .h file)

```cpp
class Name {
  private:
    members;
  public:
    members;
};
```

class member definition syntax (in a .cc file)

```cpp
returntype classname::methodname(parameters) {
  statements;
}
```

- You can name your .cc, .h file anything (unlike Java)
  - typically name them Classname.cc, Classname.h

.h file

```cpp
#ifndef _POINT_H_
#define _POINT_H_

class Point {
  public:
    Point(const int x, const int y); // constructor
    int get_x() const { return x_; } // inline member function
    int get_y() const { return y_; } // inline member function
    double Distance(const Point &p) const; // member function
    void SetLocation(const int x, const int y); // member function
  private:
    int x_; // data member
    int y_; // data member
  }; // class Point

#endif // _POINT_H_
```

Point.h

These are slightly modified versions of slides prepared by Steve O'Keeffe
.cc file

```cpp
#include <cmath>
#include "Point.h"

Point::Point(const int x, const int y) {
    x_ = x;
    this->y_ = y; // "this->" is optional, unless names conflict
}

double Point::Distance(const Point &p) const {
    // We can access p's x_ and y_ variables either through the
    // get_x(), get_y() accessor functions, or the x_, y_ private
    // member variables directly, since we're in a member
    // function of the same class.
    double distance = (x_ - p.get_x()) * (x_ - p.get_x());
    distance += (y_ - p.y_) * (y_ - p.y_);
    return sqrt(distance);
}

void Point::SetLocation(const int x, const int y) {
    x_ = x;
    y_ = y;
}
```

.usepoint.cc

Point.cc

These are slightly modified versions of slides prepared by Steve Dstinence

.cc file with main( )

```cpp
#include <iostream>
#include "Point.h"

using namespace std;

// 'main' is defined in the global name space.
// There can be only one (not one per class, as in Java).
int main(int argc, char **argv) {
    Point p1(1, 2); // stack allocate a new Point
    Point p2(4, 6); // stack allocate a new Point

    cout << "p1 is: (" << p1.get_x() << ", ");
    cout << p1.get_y() << ")" << endl;
    cout << "p2 is: (" << p2.get_x() << ", ");
    cout << p2.get_y() << ")" << endl;
    cout << "dist : " << p1.Distance(p2) << endl;
    return 0;
}
```

.usepoint.cc

These are slightly modified versions of slides prepared by Steve Dstinence
Exercise 1

Write a C++ program that:

- has a class representing a 3-dimensional point
- has the following methods:
  - return the inner product of two 3d points
  - return the distance between two 3d points
  - accessors and mutators for the x, y, z coordinates