References

Note: Arrow points to *next* instruction.

- A **reference** is an alias for another variable
  - *Alias*: another name that is bound to the aliased variable
    - Mutating a reference ***is*** mutating the aliased variable
  - Introduced in C++ as part of the language

```
int main(int argc, char** argv) {
  int x = 5, y = 10;
  int& z = x;

  z += 1;
  x += 1;

  z  = y;
  z += 1;

  return EXIT_SUCCESS;
}
```

When we use '&' in a type declaration, it is a reference.

&var still is "address of var"

x	5

y	10

reference.cc

Pass-By-Reference

Note: Arrow points to *next* instruction.

- C++ allows you to use real 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!

```
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;
}
```

Parameters are attached To variables provided by caller

(main) a	5

(main) b	10

passbyreference.cc

Poll Everywhere

pollev.com/cse33320su

- What will happen when we run this?

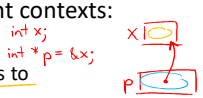
A. Output "(1,2,3)"
B. Output "(3,2,3)"
C. Compiler error about arguments to foo (in main)
D. Compiler error about body of foo
E. We're lost…

poll1.cc

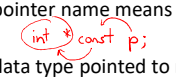
```
void foo(int& x, int* y, int z) {
  z = *y;
  x += 2;
  y = &x;
}

int main(int argc, char** argv) {
  int a = 1;
  int b = 2;
  int& c = a;

  foo(a, &b, c);
  std::cout << "(" << a << ", " << b
   << ", " << c << ")" << std::endl;

  return EXIT_SUCCESS;
}
```


`const` and Pointers

- Pointers can change data in two different contexts:
  1) You can change the value of the pointer
  2) You can change the thing the pointer points to (via dereference)



- `const` can be used to prevent either/both of these behaviors!
  - `const` next to pointer name means you can't change the value of the pointer



  - `const` next to data type pointed to means you can't use this pointer to change the thing being pointed to

  - Tip: read variable declaration from *right-to-left*

Poll Everywhere

pollev.com/cse33320su

- What will happen when we try to compile and run?

poll2.cc

```
void foo(int* const x,
         int& y, int z) {
  *x += 1;
   y *= 2;
   z -= 3;
}

int main(int argc, char** argv) {
  const int a = 1;
  int b = 2, c = 3;

  foo(&a, b, c);
  std::cout << "(" << a << ", " << b
   << ", " << c << ")" << std::endl;

  return EXIT_SUCCESS;
}
```

A. Output "(2, 4, 0)"
B. Output "(2, 4, 3)"
C. Compiler error about arguments to foo (in main)
D. Compiler error about body of foo
E. We're lost…

36

When to Use References?

STYLE TIP

- A stylistic choice, not mandated by the C++ language
- Google C++ style guide suggests:
  - Input parameters:
    - Either use values (for primitive types like `int` or small structs/objects) — Avoid making unnecessary copies
    - Or use `const` references (for complex struct/object instances)
  - Output parameters:
    - Use `const` pointers — To make sure we don't change in function; also allows const & non-const arguements
      - Unchangeable pointers referencing changeable data
  - Ordering:
    - List input parameters first, then output parameters last

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

styleguide.cc

38

Class Organization

- It's a little more complex than in C when modularizing with `struct` definition:
  - Class definition is part of interface and should go in `.h` file
    - Private members still must be included in definition (!)
  - Usually put member function definitions into companion `.cc` file with implementation details
    - Common exception: setter and getter methods
  - These files can also include non-member functions that use the class
- Unlike Java, you can name files anything you want
  - Typically `Name.cc` and `Name.h` for `class Name`

41

Poll Everywhere

pollev.com/cse33320su

- What will happen when we try to compile and run?

poll3.cc

```
class Integer {
 public:
  Integer(int x) { x_ = x; }
  int GetValue() const { return x_; }
  void SetValue(int x) const { x_ = x; }
 private:
  int x_;
};

int main(int argc, char** argv) {
  const Integer i(1);
  i.SetValue(i.GetValue() + 350);
  std::cout << i.GetValue() << std::endl;

  return EXIT_SUCCESS;
}
```

A. Output "1"
B. Output "351"
C. Compiler error about violating const-ness of i (in main)
D. Compiler error about one of the member functions
E. We're lost…

46