Pointers and `const` types

CS 251 Revision 1.9

`*` (dereference operator)
1 Argument: An r-value of a pointer type `T *`.
State change: None.
Return: The location of type `T` whose address is *arg1*. (This return value may be used as either an l-value or an r-value.)

`&` (address operator)
1 Argument: An l-value of any type `T`.
State change: None.
Return: An r-value of type `T *` that is the address of *arg1*.

Three meanings for `*` in C++

- Multiplication operator, e.g., `i = 6*x;`.
- Pointer type name, e.g., `char * str;`
- Dereferencing operator, e.g., `*ptr` where `ptr` is a variable of pointer type.

Three meanings for `&` in C++

- Binary AND operator, e.g., `i = 2&x;` (copies the next-to-last bit of x).
- Reference type name, e.g., `char &str;`
- Address operator, e.g., `&x` where `x` is a variable.

(Also: `&&` is the *logical* AND operator, e.g., `x < 3 && y == 2`

Four meanings for `const` in C++

- In a variable definition: **The value in that memory location may not be changed** using that variable name. Example:

```
const float pi = 3.14159;

pi = 3.14159265;  /* ERROR: pi has type const float, so value of pi may not be changed! */
```

- In a pointer or reference argument type: **The value(s) pointed to/referred to may not be changed** using that argument name. Example:

```
int strmod(const char * str) {
  str[0] = 'x';  /* ERROR:  str has type const char *, so the
                    characters that str points to may not be changed! */
}
```

- In a return value type: **The returned location's value may not be changed** using that return value. Example:

```
const char * second(const char *str) { return &str[1]; }
char *chptr;

chptr = second("hello"); /* ERROR: the return value for second() has type
                      const char *, so that return value may not be
                      changed; but chptr does not have a const type! */
```

- `const` method: **the method is safe to call for `const` objects**. Example:

```
class Dog {
protected:
  char *name;
  int age;
public:
  char get_initial(void) const { return name[0]; }
  void birthday() { age++; }
  ...
};

int main() {
  const Dog fido("Fido", 3);

  char ch = fido.get_initial();  // no error: get_initial is a const method
  fido.birthday();  /* ERROR because fido is a const object and
                       birthday is not a const method */
}
```