• Review

• Copy constructor and assignment
  • What is the prototype for the copy constructor?
  • What is the difference between shallow copy and deep copy?
  • Which copy does the default do?
  • Prototype of the assignment operator
  • What is the main difference between assignment operator and copy constructor?

• In Assignment No. 6, in overloading the + operator, what is wrong with the prototype:
  • PlayList operator+(PlayList & p, const song & s);
String and the [] and & operators
C-strings

• Recall that a C-string is implemented as a NULL terminated array of type `char`

```cpp
char buffer[5];
strcpy(buffer,"hi!\n");
cout<< buffer;
```

• When we use "" the compiler makes a NULL terminated const char array and fills it with the characters the programmer chose

• NOT every char array is a c-string, only those that are NULL terminated

  • Link to c-string review:
C-string and c++

- We have some features in the standard C++ libraries available to help us work more easily with C-style strings
  - The `<cstring>` library
    - Contains functions for common string operations, such as copy, compare, concatenate, length, search, tokenization, and more
      - `strlen()`, `strcpy()`, `strncpy()`, `strcat()`, `strncat()`, `strcmp()`, `strncmp()`, `strstr()`, `strtok()`
  - Special features in `<iostream>`:
    - Special built-in functions for I/O handling of C-style strings, like the insertion and extraction operators, `get()`, `getline()`, etc
    - `char str1[40];`
    - `cout << str1;`  // insertion operator for strings
    - `cin >> str1;`   // extraction, reads up to white space
    - `cin.get(str1, 40, ',');`  // reads to delimiter (comma)
    - `cin.getline(sr1, 40);`  // reads to delimiter (default delimiter is newline), discards delimiter
The Downside of C-strings

- Fixed length (when declared as static array)
- String name acts like a pointer
- Array bounds are not automatically enforced
- Must use cumbersome functions instead of intuitive operators
  - strcpy(str1, str2); instead of str1 = str2;
  - (strcmp(str1, str2)) instead of (str1 == str2)
  - strcat(str1, str2) instead of str1 += str2;
- The NULL char can be tricky
  - See sample2.cpp, sample3.cpp, sample4.cpp
String Wish List

- We would like a more intuitive string interface
  - `str1 + str2` //concatenation
  - `str1 == str2` //compare str1 and str2
  - `str1 = “Hello!\n”` //store “hello!\n” in str1
- We would like to keep some of the legacy functionality
  - `str1[4]` // returns 4\textsuperscript{th} char in str1
  - `str1[4] = 'a'` //sets 4\textsuperscript{th} char in str1 to 'a'
  - `&str1` returns the c-string (starting address) for str1
- The next programming assignment to be discussed in a while.
Overloading based on L-value and R-value

- An expression such as an array element may happen in the left hand side (lhs) or right hand side (rhs) of an assignment statement.
  - E.g. \( x = a[2]; \quad a[2] = x; \)

- When the expression in the right hand side (\(x=a[2];\)): what does the computer needs to know about the expression in order to do the assignment?  --- The value

- When the expression in the right hand side (\(a[2] = x;\)), the computer needs to know the memory location of \(a[2]\) (not the value of \(a[2]\))

- L-value of \(a\) is the reference of the variable (or expression)

- R-value of a variable is the value of the variable.
Overloading based on L-value and R-value

- Since L-value and R-value are different, C++ allows for overloading operators based on L-value and R-value.
  - Different functions are invoked depending on whether the operator happens in the left hand side or right hand side of an assignment.

```cpp
Class Someclass {
    public:
        int & operatorX(); // invoked in l-value invocation
        const int & operatorX(); // used in r-value invocation.
    ...
};
```
Overloading operator[]

- Usually done with two MEMBER functions
- Format: returntype operator[] (indextype index) const
  returntype& operator[](indextype index)
- The const member function allows us to read the element from a const object
- The non-const member function returns a reference to the element that can be modified
- See sample5.cpp
Overloading the address operator

- The address operator can be overloaded just like any other operator (sample6.cpp)