CMPE 180-92

Data Structures and Algorithms in C++

October 5 Class Meeting

Department of Computer Engineering
San Jose State University

Fall 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

SILICON VALLEY'S FIRST CHOICE FOR NEW ENGINEERING HIRES
— Silicon Valley Business Journal, 2013

Assignment #6 Sample Solution

Book.h

```
class Book
{
public:
    /**
     * Book categories.
     */
    enum class Category { FICTION, HISTORY, TECHNICAL, NONE };

    /**
     * Default constructor.
     */
    Book();

    /**
     * Constructor.
     */
    Book(string isbn, string last, string first, string title, Category category);

    /**
     * Destructor.
     */
    ~Book();
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
/**
 * Getter.
 * @return the book's ISBN.
 */
string get_isbn() const;

/**
 * Getter.
 * @return the author's last name.
 */
string get_last() const;

/**
 * Getter.
 * @return the author's first name.
 */
string get_first() const;
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
/**
 * Getter.
 * @return the book's title.
 */
string get_title() const;

/**
 * Getter.
 * @return the book's category.
 */
Category get_category() const;
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
/**
 * Overloaded input stream extraction operator for a book.
 * Reads from a CSV file.
 * @param istream the input stream.
 * @param book the book to input.
 * @return the input stream.
 */
friend istream& operator >>(istream& ins, Book& emp);

/**
 * Overloaded output stream insertion operator for a book.
 * @param ostream the output stream.
 * @param book the book to output.
 * @return the output stream.
 */
friend ostream& operator <<(ostream& outs, const Book& emp);
```

Assignment #6 Sample Solution, *cont'd*

Book.h

```
private:
    string isbn;          // ISBN
    string last;          // author's last name
    string first;         // author's first name
    string title;         // book title
    Category category;    // book category
};


/**
 * Overloaded output stream insertion operator for a book category.
 * Doesn't need to be a friend since it doesn't access any
 * private members.
 * @param ostream the output stream.
 * @param book the category to output.
 * @return the output stream.
 */
ostream& operator <<(ostream& outs, const Book::Category& category);
```

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <stdio.h>
#include "Book.h"

using namespace std;

Book::Book()
    : isbn(""), last(""), first(""), title(""),
      category(Category::NONE)
{}

Book::Book(string isbn, string last, string first, string title,
           Category category)
    : isbn(isbn), last(last), first(first), title(title),
      category(category)
{}

Book::~Book()
{}
```

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
string Book::get_isbn()  const { return isbn; }
string Book::get_last()  const { return last; }
string Book::get_first() const { return first; }
string Book::get_title() const { return title; }

Book::Category Book::get_category() const { return category; }
```

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
istream& operator >>(istream& ins, Book& book)
{
    ins.get();  // skip the blank after the command

    getline(ins, book.isbn,  ',');
    getline(ins, book.last,  ',');
    getline(ins, book.first, ',');
    getline(ins, book.title, ',');

    string catstr;
    getline(ins, catstr);

    book.category = Book::Category::NONE;

    if      (catstr == "fiction")   book.category = Book::Category::FICTION;
    else if (catstr == "history")   book.category = Book::Category::HISTORY;
    else if (catstr == "technical") book.category = Book::Category::TECHNICAL;

    return ins;
}
```

Assignment #6 Sample Solution, *cont'd*

```
ostream& operator <<(ostream& outs, const Book::Category& category)
{
    switch (category)
    {
        case Book::Category::FICTION:   outs << "fiction";      break;
        case Book::Category::HISTORY:   outs << "history";      break;
        case Book::Category::TECHNICAL: outs << "technical";    break;
        case Book::Category::NONE:      outs << "none";         break;
    }

    return outs;
}
```

Book.cpp

Assignment #6 Sample Solution, *cont'd*

Book.cpp

```
ostream& operator <<(ostream& outs, const Book& book)
{
    outs << "Book{ISBN=" << book.isbn << ", last=" << book.last
         << ", first=" << book.first << ", title=" << book.title
         << ", category=" << book.category << "}";
    return outs;
}
```

BookApp.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iomanip>
#include "Book.h"

using namespace std;

// Status codes.
enum class StatusCode {OK, DUPLICATE, NOT_FOUND, INVALID_COMMAND};

/**
 * Execute a command.
 * @param command the command.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 */
StatusCode execute(const char command, istream &input,
                   vector<Book>& catalog);
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * Insert a new book into the catalog at the appropriate position
 * to maintain sort order by ISBN.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @param index set to the catalog index of the new record.
 * @return the status code of this operation.
 */
StatusCode insert(istream &input, vector<Book>& catalog, int &index);

/**
 * Remove a book from the catalog.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @param book set to the removed book.
 * @return the status code of this operation.
 */
StatusCode remove(istream &input, vector<Book>& catalog, Book& book);
```

BookApp.cpp

```
/**
 * Match books.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @return a vector of the indices of the matching books.
 */
vector<int> match(istream &input, vector<Book>& catalog);

/**
 * Match the book in the catalog with the given ISBN.
 * @param istream the input data stream.
 * @param catalog the vector of book records.
 * @return a vector of the index of the matching book.
 */
vector<int> match_by_isbn(const string last,
                          const vector<Book>& catalog);
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * Match the books in the catalog with the given author's last name.
 * Use a linear search.
 * @param last the author's last name.
 * @param catalog the book vector.
 * @return a vector of the indices of the matching books.
 */
vector<int> match_by_author(const string last,
                            const vector<Book>& catalog);

/**
 * Match the books in the catalog in the given category.
 * Use a linear search.
 * @param catstr the category.
 * @param catalog the book vector.
 * @return a vector of the indices of the matching books.
 */
vector<int> match_by_category(string catstr,
                              const vector<Book>& catalog);
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * Match all the books in the catalog.
 * Use a linear search.
 * @param last the author's last name.
 * @param catalog the book vector.
 * @return a vector of the indices of the matching books.
 */
vector<int> match_all(const vector<Book>& catalog);

/**
 * Process an invalid command.
 * @param istream the input data stream.
 * @return the status code.
 */
StatusCode invalid_command(istream &input);
```

BookApp.cpp

```
/**
 * Find the book in the catalog with the given ISBN.
 * Use a binary search.
 * @param isbn the ISBN.
 * @param catalog the vector of book records.
 * @return the vector index of the book if found, else return -1.
 */
int find(const string isbn, const vector<Book>& catalog);

/**
 * Print an error message.
 * @param status the status code.
 */
void print_error_message(StatusCode status);

const string INPUT_FILE_NAME = "commands.in";
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
/**
 * The main. Open the command input file and loop to process commands.
 */
int main()
{
    // Open the input file.
    ifstream input;
    input.open(INPUT_FILE_NAME);
    if (input.fail())
    {
        cout << "Failed to open " << INPUT_FILE_NAME << endl;
        return -1;
    }

    vector<Book> catalog;  // book catalog

    char command;
    input >> command;  // read the first command
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
    /**
     * Loop to read commands until the end of file.
     */
    while (!input.fail())
    {
        cout << endl << command << "  ";

        StatusCode status = execute(command, input, catalog);
        if (status != StatusCode::OK) print_error_message(status);

        input >> command;
    }

    return 0;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
StatusCode execute(const char command, istream &input,
                   vector<Book>& catalog)
{
    int index;
    StatusCode status;
    Book book;

    // Execute the command.
    switch (command)
    {
        case '+':
            status = insert(input, catalog, index);
            book = catalog[index];
            cout << "Inserted at index " << index << ": "
                 << book << endl;
            break;

        case '-':
            status = remove(input, catalog, book);
            cout << "Removed " << book << endl;
            break;
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
        case '?':
        {
            vector<int> matches = match(input, catalog);
            for (int i : matches) cout << catalog[i] << endl;
            status = StatusCode::OK;
            break;
        }

        default:
            status = invalid_command(input);
            break;
    }

    return status;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
StatusCode insert(istream &input, vector<Book>& catalog, int& index)
{
    // Read the book information.
    Book book;
    input >> book;

    string isbn = book.get_isbn();

    // Loop to find the proper insertion point.
    index = 0;
    while (   (index < catalog.size())
           && (isbn > catalog[index].get_isbn())) index++;
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
    // Check the insertion point.
    if (index >= catalog.size())
    {
        catalog.push_back(book);        // append at the end
        return StatusCode::OK;
    }
    else if (isbn == catalog[index].get_isbn())
    {
        return StatusCode::DUPLICATE;   // duplicate
    }
    else
    {
        catalog.insert(catalog.begin() + index, book);  // insert
        return StatusCode::OK;
    }
}
```

Assignment #6 Sample Solution, *cont'd*

```
StatusCode remove(istream &input, vector<Book>& catalog, Book& book)
{
    string isbn;
    input >> isbn;

    // Look for the book record with a matching ISBN.
    int index = find(isbn, catalog);
    if (index == -1)
    {
        book = Book(isbn, "", "", "", Book::Category::NONE);
        return StatusCode::NOT_FOUND;
    }

    // Remove the matching book from the catalog.
    book = catalog[index];
    catalog.erase(catalog.begin() + index);
    return StatusCode::OK;
}
```

BookApp.cpp

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match(istream &input, vector<Book>& catalog)
{
    vector<int> matches;

    string str;
    getline(input, str);

    if (str == "")
    {
        matches = match_all(catalog);
    }

    else if (str.find("isbn=") != str.npos)
    {
        string isbn = str.substr(str.find("=") + 1);
        matches = match_by_isbn(isbn, catalog);
    }
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
    else if (str.find("author=") != str.npos)
    {
        string last = str.substr(str.find("=") + 1);
        matches = match_by_author(last, catalog);
    }

    else if (str.find("category=") != str.npos)
    {
        string category = str.substr(str.find("=") + 1);
        matches = match_by_category(category, catalog);
    }

    return matches;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match_by_isbn(const string isbn,
                          const vector<Book>& catalog)
{
    vector<int> matches;

    cout << "Book with ISBN " << isbn << ":" << endl;

    int index = find(isbn, catalog);
    if (index != -1) matches.push_back(index);

    return matches;
}
```

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match_by_author(const string last,
                            const vector<Book>& catalog)
{
    vector<int> matches;

    cout << "Books by author " << last << ":" << endl;

    // Do a linear search.
    for (int i = 0; i < catalog.size(); i++)
    {
        Book book = catalog[i];
        if (last == book.get_last()) matches.push_back(i);
    }

    return matches;
}
```

Assignment #6 Sample Solution, *cont'd*

```
vector<int> match_by_category(string catstr, const vector<Book>& catalog)
{
    vector<int> matches;

    Book::Category category = catstr == "fiction"   ? Book::Category::FICTION
                            : catstr == "history"   ? Book::Category::HISTORY
                            : catstr == "technical" ? Book::Category::TECHNICAL
                            :                          Book::Category::NONE;


    cout << "Books in category " << category << ":" << endl;

    // Do a linear search.
    for (int i = 0; i < catalog.size(); i++)
    {
        Book book = catalog[i];
        if (category == book.get_category()) matches.push_back(i);
    }


    return matches;
}
```

BookApp.cpp

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
vector<int> match_all(const vector<Book>& catalog)
{
    vector<int> matches;

    cout << "All books in the catalog:" << endl;

    for (int i = 0; i < catalog.size(); i++) matches.push_back(i);
    return matches;
}

StatusCode invalid_command(istream &input)
{
    // Read and ignore the rest of the input line.
    string ignore;
    getline(input, ignore);

    return StatusCode::INVALID_COMMAND;
}
```

Assignment #6 Sample Solution, *cont'd*

```
int find(const string isbn, const vector<Book>& catalog)
{
    // Do a binary search.
    int low = 0;
    int high = catalog.size();

    while (low <= high)
    {
        int mid = (low + high)/2;
        Book book = catalog[mid];

        if (isbn == book.get_isbn())
        {
            return mid;     // found
        }
        else if (isbn < book.get_isbn())
        {
            high = mid - 1; // search lower half
        }
        else
        {
            low  = mid + 1; // search upper half
        }
    }

    return -1;  // not found
}
```

BookApp.cpp

Assignment #6 Sample Solution, *cont'd*

BookApp.cpp

```
void print_error_message(StatusCode status)
{
    switch (status)
    {
        case StatusCode::DUPLICATE:
            cout << "*** Duplicate ISDN ***" << endl;
            break;

        case StatusCode::NOT_FOUND:
            cout << "*** Book not found ***" << endl;
            break;

        case StatusCode::INVALID_COMMAND:
            cout << "*** Invalid command ***" << endl;
            break;

        default:  break;
    }
}
```

A “Safe” Array Type: Version 1

- We will develop a new array type that is “safe”.
  - It will allocate the array dynamically.
  - It will check all subscript values to ensure that they are in the legal range ($0 \le \text{index} < \text{array length}$).
- We’ll start with an integer array.

A “Safe” Array Type: Version 1, *cont’d*

SafeArray1.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    ~SafeArray();

    int get_length() const;

    int at(int i) const;
    void set(int i, int value);

    void operator =(const SafeArray& rhs);

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 1, *cont’d*

SafeArray1.cpp

```
SafeArray::SafeArray() : elements(nullptr), length(0)
{
}

SafeArray::SafeArray(int len) : elements(nullptr), length(len)
{
    elements = new int[length];
}

SafeArray::~SafeArray()
{
    if (elements != nullptr) delete[] elements;
}

int SafeArray::get_length() const { return length; }

int SafeArray::at(int i) const
{
    assert((i >= 0) && (i < length));
    return elements[i];
}
```

A “Safe” Array Type: Version 1, *cont’d*

SafeArray.cpp

```
void SafeArray::set(int i, int value)
{
    assert((i >= 0) && (i < length));
    elements[i] = value;
}

void SafeArray::operator =(const SafeArray& rhs)
{
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }
}
```

A “Safe” Array Type: Version 1, *cont’d*

SafeArrayTests1.cpp

```
int main()
{
    SafeArray a1(10), a2;
    //SafeArray a3;

    for (int i = 0; i < 10; i++) a1.set(i, 10*i);

    a2 = a1;
    a1.set(4, -a1.at(4));

    cout << "a1 ="; print(a1);
    cout << "a2 ="; print(a2);

    //a3 = a2 = a1;
    return 0;
}

void print(SafeArray& a)
{
    for (int i = 0; i < a.get_length(); i++) cout << " " << a.at(i);
    cout << endl;
}
```

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
```

A “Safe” Array Type: Version 1, *cont’d*

- What happens if you try to chain assignments?

```
SafeArray a1(10), a2;
SafeArray a3;

...

a3 = a2 = a1;
```

```
../SafeArrayTests.cpp:20:8: error: no viable overloaded '='
    a3 = a2 = a1;
    ~~ ^ ~~~~~~~~
../SafeArray.h:16:10: note: candidate function not viable:
  cannot convert argument of incomplete type 'void' to 'const SafeArray'
    void operator =(const SafeArray& rhs);
         ^
1 error generated.
```

A “Safe” Array Type: Version 2

SafeArray2.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    ~SafeArray();

    int get_length() const;

    int at(int i) const;
    void set(int i, int value);

    SafeArray& operator =(const SafeArray& rhs);

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 2, *cont’d*

```
SafeArray& SafeArray::operator =(const SafeArray& rhs)
{
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }

    return *this;
}
```

SafeArray2.cpp

A “Safe” Array Type: Version 2, *cont’d*

SafeArrayTests2.cpp

```
int main()
{
    SafeArray a1(10), a2, a3;

    for (int i = 0; i < 10; i++) a1.set(i, 10*i);

    a3 = a2 = a1;
    a1.set(4, -a1.at(4));

    cout << "a1 ="; print(a1);
    cout << "a2 ="; print(a2);
    cout << "a3 ="; print(a3);

    return 0;
}
```

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
a3 = 0 10 20 30 40 50 60 70 80 90
```

A “Safe” Array Type: Version 2, *cont’d*

- What happens the program executes

```
a1 = a1;
```

```
SafeArray& SafeArray::operator =(const SafeArray& rhs)
{
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }

    return *this;
}
```

SafeArray2.cpp

A “Safe” Array Type: Version 3

- The solution:

```
SafeArray& SafeArray::operator =(const SafeArray& rhs)
{
    if (this == &rhs) return *this;
    if (elements != nullptr) delete[] elements;

    length = rhs.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = rhs.elements[i];
    }

    return *this;
}
```

SafeArray3.cpp

Break

A “Safe” Array Type: Version 4

- The `at` and `set` member functions are awkward to use.

- Why can’t we use subscripts on a smart array as if it were a regular array?

- We can overload the subscript operator `[]`
  - We want the subscripts to be usable on <u>either side</u> of an assignment.
  - Example:

```
a1[4] = -a1[4];
```

A "Safe" Array Type: Version 4, *cont'd*

SafeArray4.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    ~SafeArray();

    int get_length() const;

    int at(int i) const;
    void set(int i, int value);

    SafeArray& operator =(const SafeArray& rhs);
    int& operator [](int i) const;

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 4, *cont’d*

```
int& SafeArray::operator [](int i) const
{
    assert((i >= 0) && (i < length));
    return elements[i];
}
```

SafeArray4.cpp

A “Safe” Array Type: Version 4, *cont’d*

SafeArrayTests4.cpp

```
int main()
{
    SafeArray a1(10), a2, a3;

    for (int i = 0; i < 10; i++) a1[i] = 10*i;

    a3 = a2 = a1;
    a1[4] = -a1[4];

    cout << "a1 ="; print(a1);
    cout << "a2 ="; print(a2);
    cout << "a3 ="; print(a3);

    return 0;
}

void print(SafeArray& a)
{
    for (int i = 0; i < a.get_length(); i++) cout << " " << a[i];
    cout << endl;
}
```

```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
a3 = 0 10 20 30 40 50 60 70 80 90
```

A “Safe” Array Type: Version 4, *cont’d*

- What if we passed the smart array object by value instead of by reference?

```
void print(SafeArray a)
{
    for (int i = 0; i < a.get_length(); i++)
    {
        cout << " " << a[i];
    }
    cout << endl;
}
```

SafeArrayTests4.cpp

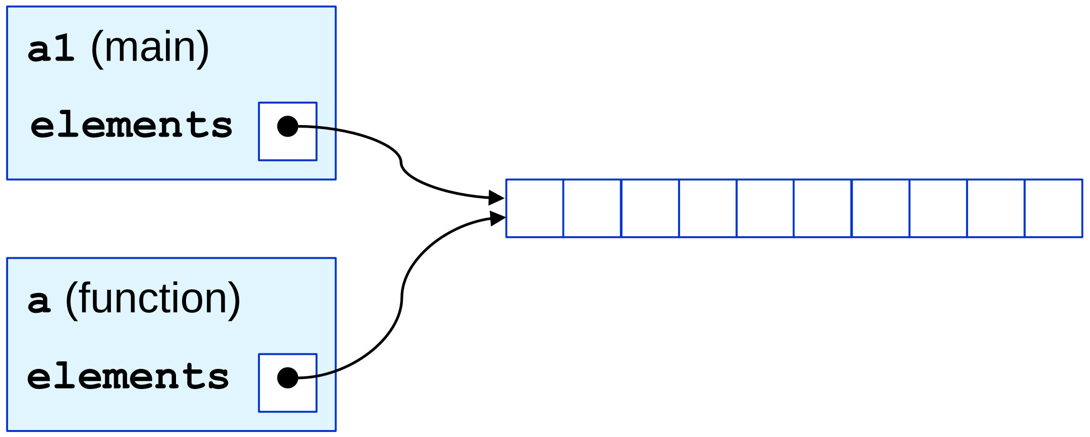
```
a1 = 0 10 20 30 -40 50 60 70 80 90
a2 = 0 10 20 30 40 50 60 70 80 90
a3 = 0 10 20 30 40 50 60 70 80 90
SafeArray4(78650,0x7fffbe7fe3c0) malloc: *** error for object 0x7fbe64c02740:
                                         pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
```

A “Safe” Array Type: Version 5

- A very unexpected side effect!
- At the end, the program attempted to delete the private dynamic array elements.
- But the dynamic array was already deleted by the destructor.
  - So who tried to delete the array again?
- Why did passing a SmartArray object by value instead of by reference to the print function cause this problem?

A "Safe" Array Type: Version 5, *cont'd*

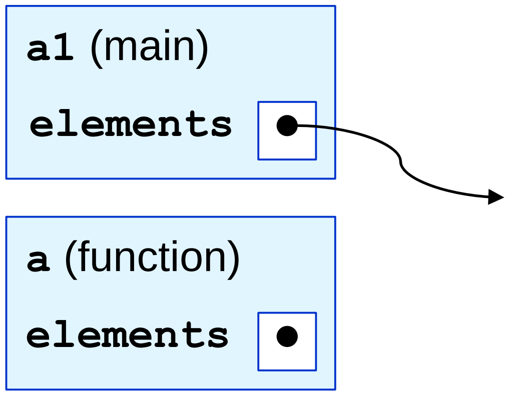
- When a `SmartArray` object is passed by value to the `print` function, a copy is made.
- This copy will point to the same dynamic array.
  - This is what the default copy constructor does.



A “Safe” Array Type: Version 5, *cont’d*

- When the print function completes and returns, its local variables go out of scope.

- The `SmartArray` object’s destructor is called, which deletes the dynamic array.
  - Now variable `a1` has a dangling pointer.
  - When the program is ready to terminate, it calls `a1`’s destructor.
  - An error occurs because of the attempt to delete memory that has already been deleted.



Copy Constructor

- Every class has a copy constructor.
  - C++ supplies a <u>default</u> copy constructor.
  - It may not do what you want, so you can write one.

- A copy constructor has only <u>one</u> parameter, which is a <u>reference to the same class</u>.

- A copy constructor is called when:
  - A new object is created and initialized using another object of the same type.
  - An object is passed by value to a function.
  - An object is returned by a function.

A “Safe” Array Type: Version 5, *cont’d*

SafeArray5.h

```
class SafeArray
{
public:
    SafeArray();
    SafeArray(int len);
    SafeArray(const SafeArray& other);  // copy constructor
    ~SafeArray();

    int get_length() const;

    SafeArray& operator =(const SafeArray& rhs);
    int& operator [](int i) const;

private:
    int *elements;
    int length;
};
```

A “Safe” Array Type: Version 5, *cont’d*

```
SafeArray::SafeArray(const SafeArray& other)
    : elements(nullptr), length(0)
{
    length = other.length;
    elements = new int[length];

    for (int i = 0; i < length; i++)
    {
        elements[i] = other.elements[i];
    }
}
```

SafeArray5.cpp

- Now the copy of the object has a separate copy of the contents of the **`elements`** array.

Shorthand for Pointer Expressions

```
class Node
{
public:
    Node(int value);
    ~Node();

    int data;
    Node *next;
};
```

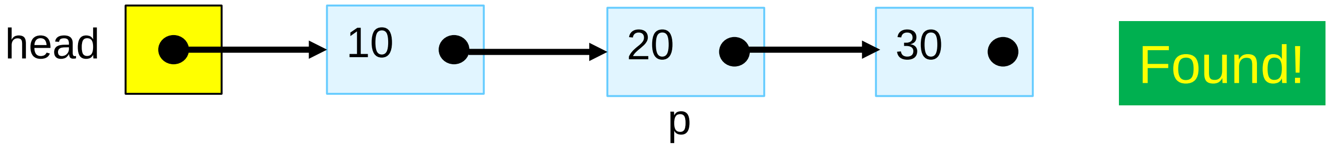
```
Node *head;
```

- The expression `head->data` is the preferred shorthand for `(*head).data`

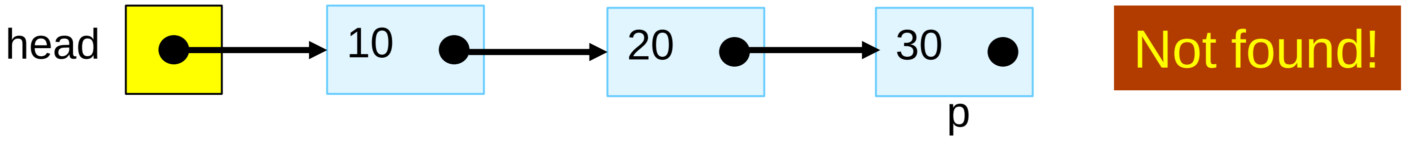
Searching a Sorted Linked List

```
Node *SortedLinkedList::find(int value) const
{
    Node *p = head;

    // Search the sorted list.
    while ((p != nullptr) && (value > p->data)) p = p->next;

    if ((p != nullptr) && (value == p->data)) return p;        // found
    else                                       return nullptr;  // not found
}
```

- Search for 20:



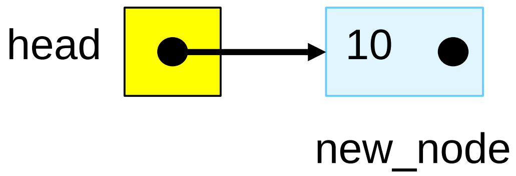
- Search for 25:



Inserting into a Sorted Linked List

- Insert the first element into a sorted linked list.

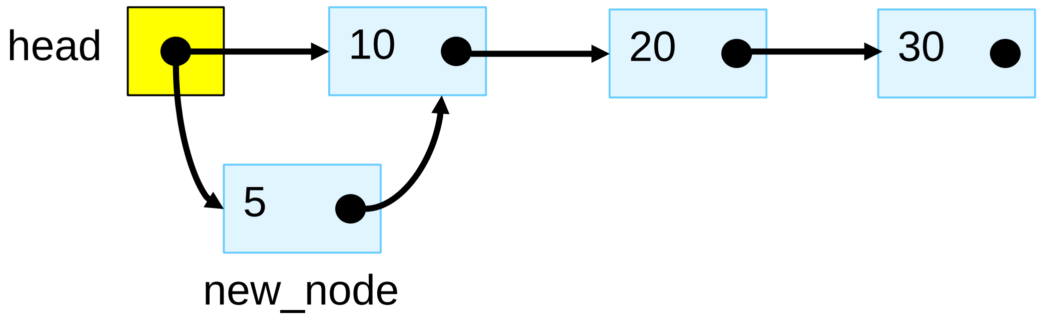
```
if (head == nullptr)
{
    head = new_node;
    return new_node;
}
```



Inserting into a Sorted Linked List, *cont'd*

- Insert at the beginning of an existing sorted linked list.

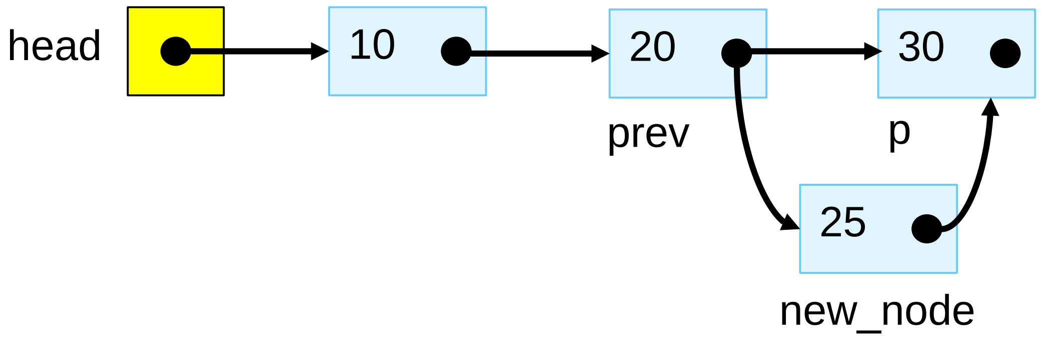
```
else if (value < head->data)
{
    new_node->next = head;
    head = new_node;
    return new_node;
}
```



Inserting into a Sorted Linked List, *cont'd*

- Insert into the middle of a sorted linked list.

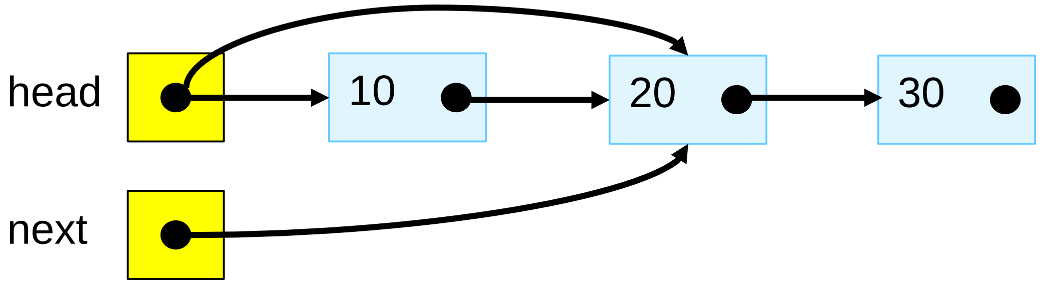
```
while ((p != nullptr) && (value >= p->data))
{
    prev = p;
    p = p->next;
}

prev->next = new_node;
new_node->next = p;
return new_node;
```



Removing from a Sorted Linked List

- Remove from the head of a sorted list.

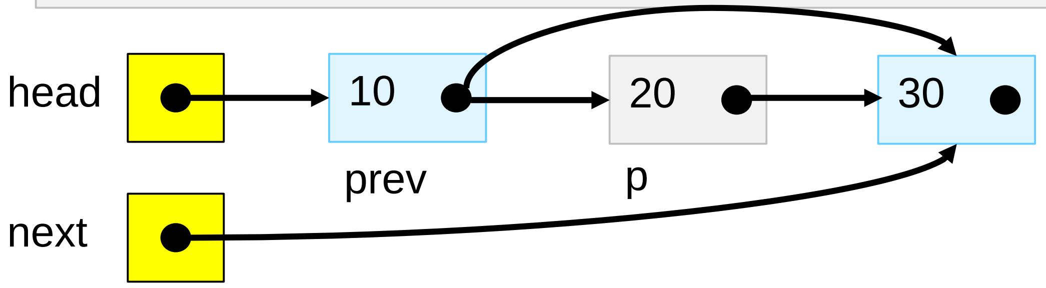
```
if (value == head->data)
{
    Node *next = head->next;
    delete head;
    head = next;
    return;
}
```



Removing from a Sorted Linked List, *cont'd*

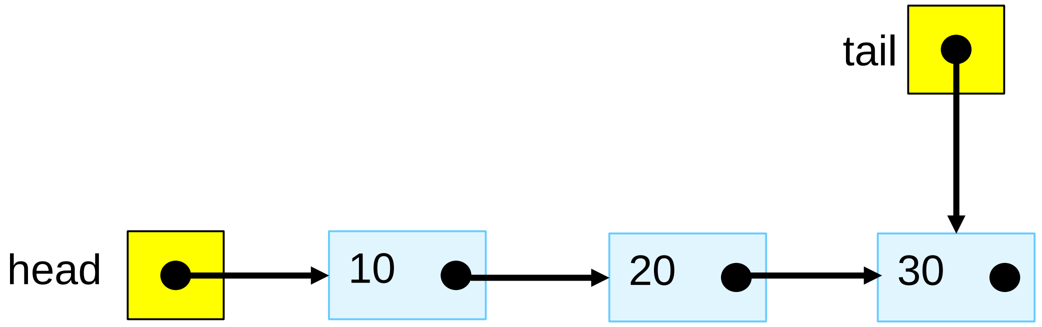
- Remove from the middle of a sorted list.

```
while ((p != nullptr) && (value > p->data))
{
    prev = p;
    p = p->next;
}

if ((p != nullptr) && (value == p->data))
{
    Node *next = p->next;
    delete p;
    prev->next = next;
}
```



Linked List Tail

- Often there are advantages for a linked list to maintain both a head pointer and a tail pointer.

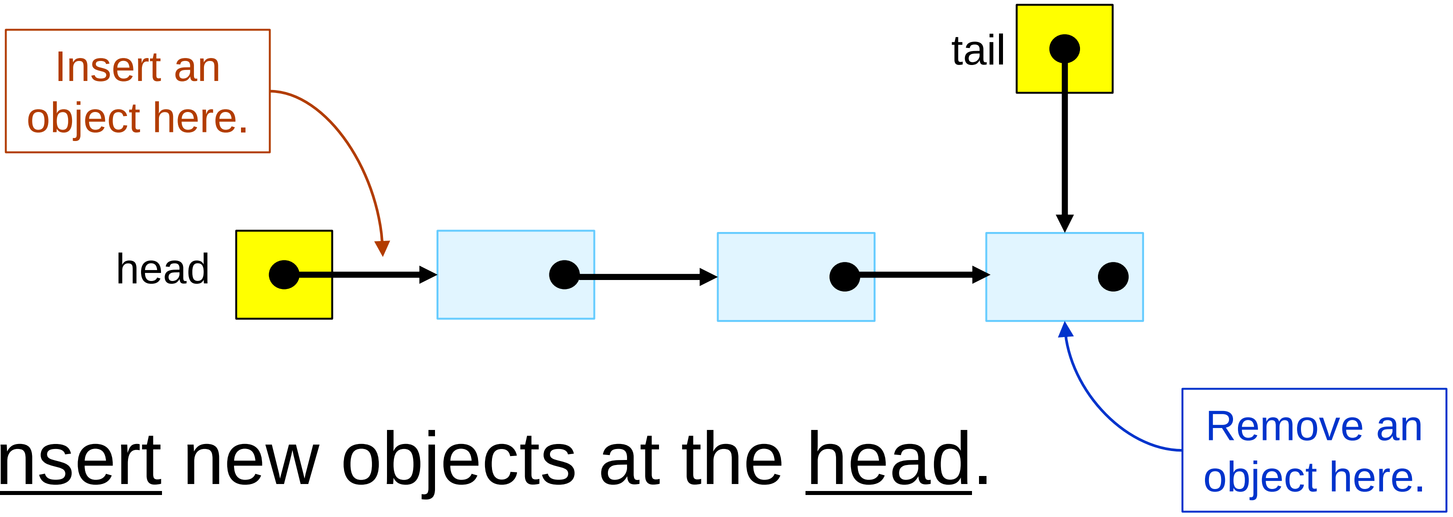


Queue

- A queue is a data structure which you can insert objects into and from which you can remove objects.
- The queue maintains the <u>order</u> that the objects are inserted.
- Objects are removed from the queue <u>in the same order</u> that they were inserted.
- This is commonly known as first-in first-out (FIFO).

Queue, *cont'd*

- We can use a linked list to implement a queue.



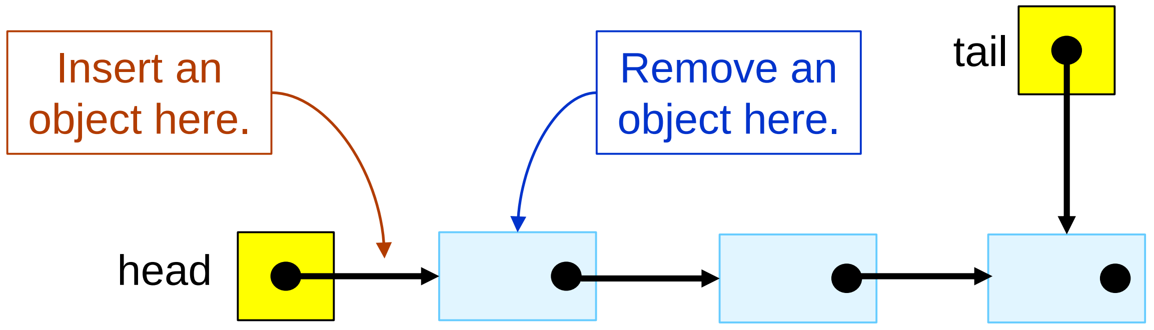
- Insert new objects at the head.
- Remove objects at the tail.
- Objects in the queue are in arrival order.
  - Not necessary for the objects to be in data order.

Stack

- A stack is a data structure into which you can insert objects and from which you can remove objects.
- The stack maintains the order that the objects are inserted.
- Objects are removed from the stack in the reverse order that they were inserted.
- This is commonly known as last-in first-out (LIFO).

Stack, *cont'd*

- We can use a linked list to implement a stack.



- Insert (push) new objects at the head.
- Remove (pop) objects at the head.

Midterm Next Week

- Combination of multiple-choice, short answer, and short programming (such as a function or a class declaration).
- Covers
  - all lectures through today
  - Savitch book chapters 1 – 13
  - assignments 1 – 7
- Closed book and laptop
- 75 minutes

Assignment #7

- Practice with linked lists.
  - Write-up and data files in Canvas by Friday.

- <u>Read from text files</u> containing data about books by various authors.
  - Each book has an ISBN, its author's last and first names, and the book title.
  - Each text file contains books from one category, already sorted by ISBN.
- <u>Create separate linked lists</u> of books from each category; i.e., a linked list per input text file.

Assignment #7, *cont'd*

- Print each category list of books.
- Merge all the separate category lists into a single book list, sorted by ISBN.
- Print the merged list.
- Split the merged list into two sublists, one sublist for authors with last names starting with A – M and the second sublist for authors with last names starting with N – Z,
- Print the two sublists.