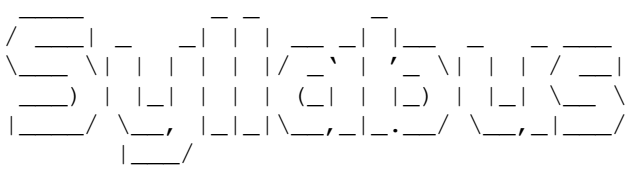


Syllabus: Programming and Algorithms I

Catalog Description

Prerequisites: At least one year of high school algebra and strong computer skills or CSCI 101.
A first-semester object-oriented programming course, providing an overview of computer systems and an introduction to problem solving, object-oriented software design, and programming. Coverage includes the software life cycle, as well as algorithms and their role in software design. Students are expected to design, implement, and test a number of programs. 3 hours lecture, 2 hours activity.

Objectives

* Develop and debug programs in C and C++
* Use proper programming style and software design techniques
* Use proper documentation techniques
* Use software development tools to develop software
* Use the concepts of object-oriented and structured programming
* Develop a working knowledge of a Unix-based operating system.

Outcomes

* Design software products
* Write applications in C and C++ with
    * Procedural and Object Oriented design principles
    * sequential execution of code
    * conditionals
    * looping constructs (iteration)
    * classes from class libraries
    * user-defined classes
    * primitive data types, reference variables, and arrays
    * use a development environment to develop and debug applications
* Document programs so that others can understand their operation
* Test programs to verify their operation

Meeting Times

Lectures (Section 5)
Monday 1:00p - 1:50p in Siskiyou Hall 120
Wednesday 1:00p - 1:50p in Siskiyou Hall 120
Friday 1:00p - 1:50p in Siskiyou Hall 120

Lab Section 6
Monday 4:00p - 4:50p in Oâ\200\231Connell 251
Wednesday 4:00p - 4:50p in Oâ\200\231Connell 251

Lab Section 7
Tuesday 4:00p - 4:50p in Oâ\200\231Connell 251
Thursday 4:00p - 4:50p in Oâ\200\231Connell 251

Instructor

Name: Todd A. Gibson
Email: tgibson@augustcouncil.com
Course Web Page: http://augustcouncil.com/~tgibson/chico/csci111/
Office: Oâ\200\231Connell 222
Office Hours: Monday, 2:00p - 4:00p and Tuesday, 1:00p - 4:00p

Outside of class and office hours, email is the best way to get in touch with me. In all emails, please structure your subject as "CSCI 111: (your subject here)".

Textbook

Title: Problem Solving with C++, 8th Edition (7th Edition is ok) by Walter Savitch

Accreditation Category Content

This course embodies a significant portion of
* (a) Theoretical Foundations
* (b) Problem Analysis and
* (c) Solutions Design

Relationship of Course to Program Objectives

This course supports the achievement of the following program objectives:
* All students will be able to analyze and solve computing problems, or problems in related areas, and to continually upgrade their knowledge and skills.
* All students will be effective oral and written communicators and be able to function effectively as members of multi-disciplinary teams.
* Those graduates who pursue careers as computing professionals will have the skills to use and design new and innovative systems that meet society's needs.
* Those graduates who pursue advanced degrees will have the skills to succeed in graduate programs in computing and related fields.

Course format

The goal of the course is to keep your hands on the keyboard as much as possible. Although there will be readings assigned from the text for guidance, the goal is to present programming problems which drive you to the textbook for answers. Time during lectures will not only be spent discussing new material, but will also be used to model object-oriented problem-solving techniques.

Grading

40%	Major deliverables
30%	Minor deliverables
15%	Midterm exam
15%	Final exam

Grading is not on a curve. 90-100 = A, 80-89 = B, etc. Those at the borders of a grade range (e.g., 89 or 90) will have a + or - appended to their grade.

Course Topics

* Programming Concepts
    * Software Development Life Cycle
    * Programming Development Tools (CLI and IDE Ã¢** Proficiency in one, exposure to boths
    * Coding Standards
* Variables and Data Types
    * Assignment Statements
    * Arithmetic Operations
    * Scalar and Vector Data Types
    * User Defined Types
* Input/Output Mechanisms
    * Console/Stream Interaction
    * File Operations
* Programming Constructs
    * Conditionals
    * Looping
* Functions, Parameter Passing, Variable Scop
* Classes & Object Oriented Concept

        * Classes and Class librarie
        * User defined Classe
        * Object Oriented Desig
    * Pointers, Linked Lists, Dynamic Allocation (at a conceptual level
    * Operating System Interactio
        * Standard Library Function

Exams

No late exams will be given unless arrangements are made with the instructor PRIOR to the exam date. Exams will cover material from the textbook, lectures, labs, and various deliverables. The fundamental nature of this course requires that all exams be cumulative, although the focus of the exams will be on the untested topics.

Cheating & Academic Honesty

Any cheating will result in an immediate F in the course and notification to the University.

Dr. Henry has posted a thorough summary of what constitutes cheating in computer science at http://www.ecst.csuchico.edu/~tyson/classes/general/cheating.html.

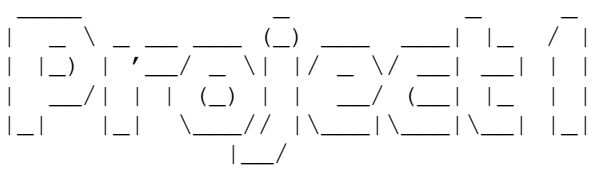
The University has posted the academic integrity policy at http://www.csuchico.edu/prs/EMs/2004/04-036.shtml. In this Executive Memorandum 04-036, President Zingg uses the following definition of "academic integrity":

> a commitment, even in the face of adversity, to five fundamental values: honesty, trust, fairness, respect, and responsibility. â\200\223Center for Academic Integrity. â\200\234Fundamental Values Project.â\200\235
23
> March 2004
> http://www.academicintegrity.org/fundamental_values_project/index.php

In the same memorandum, he uses the following definition of â\200\234cheatingâ\200\235:

> Cheating is intentional fraud or deception for the purpose of improving a grade or obtaining course credit and includes all behavior intended to gain unearned academic advantage. Cheating includes either helping or attempting to help another person cheat. â\200\223Adapted from the Office of Student Judicial Affairs at California State University, Chico. â\200\234Cheating, an Ounce of Prevention.â\200\235

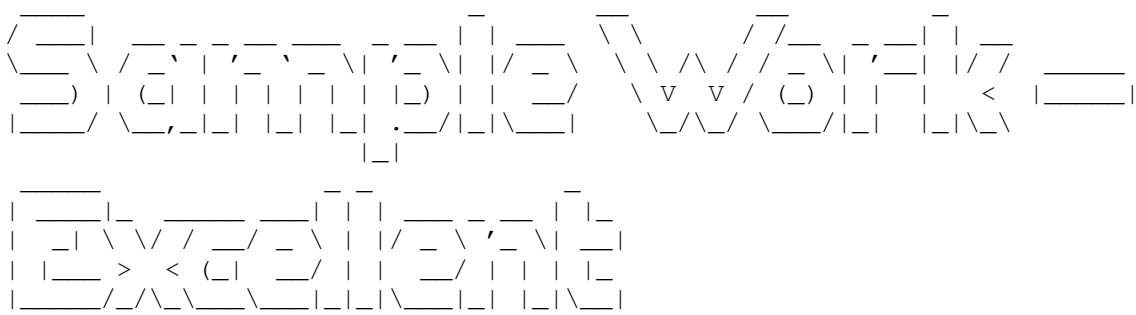
If there are any questions about what constitutes cheating, please do not hesitate to ask. If it feels like it could be a gray area, ask me. It can be tricky what you are allowed to discuss and share with fellow classmates, and what you cannot. Better to ask and be safe!



Library Gradesheet, Advanced C++

#	**Points Possible**	**Points Rec'd**	**Section**
1.	**5**		Program compiles successfully.
2.	**5**		Program runs correctly as specified in the requirements.
3.	**4**		Film, video, book, and periodical are classes ultimately derived from the same base class.
4.	**5**		The different media types don't read from files and don't manage their own list.
5.	**5**		Test program can initialize card catalog and ask user for one of four searches to perform.
6.	**4**		Card catalog can perform four different searches.
7.	**5**		Card catalog returns list of items found in search.
8.	**5**		Test program prints list of matches returned by card catalog.
9.	**5**		Card catalog either has or communicates with a list.
10.	**5**		Items are read from files only once at program start up.
11.	**5**		Items are successfully read from file.
12.	**4**		Items read from file are placed in list.
13.	**4**		Code is appropriately broken into .cpp and .h files. (coding standard #7)
14.	**5**		Memory allocated with new is properly released with delete (coding standard #6).
15.	**3**		Constructors are used (coding standard #1)
16.	**4**		Opened files are explicitly closed (coding standard #10).
17.	**7**		Polymorphism is properly used in searching and printing algorithms.
18.	**6**		Proper encapsulation and abstraction.
19.	**7**		Constants: used properly for function arguments, return types, and member functions
20.	**7**		References and pointers: used properly for function arguments and return types.
Total	**100**		

```
From pthompson@computerwerkerz.com  Mon Jan 9 14:55:05 2012
Return-Path: <pthompson@computerwerkerz.com>
Received: from onecomp.computerwerkerz.com
  (onecomp.computerwerkerz.com [10.0.1.1])
    by redcomp.computerwerkerz.com (8.11.6/8.11.2) with ESMTP id g3EMt5T26130
    for <ynhere@redcomp.computerwerkerz.com>; Mon, 9 Jan 2012 14:55:05 -0700
Message-ID: <9A815851ACA4F54395A169F2DOEK35E60384F6@ES2.computerwerkerz.com>
From: Patricia Thompson <pthompson@computerwerkerz.com>
To: "Your Name Here (E-mail)" <ynhere@computerwerkerz.com>
Subject: Work overflow
Date: Mon, 9 Jan 2012 13:52:40 -0800
MIME-Version: 1.0
X-Mailer: Internet Mail Service (5.15.6883.19)
Content-Type: text/plain;
        charset="iso-8859-1"

Hello!
The rumors are true!  The senior engineers are swamped on the Shoppy-Chops
Grocery project, so we need to pull you out of tech support and put you
on some programming tasks.

The Web administrator is currently upgrading some core internals for the
Web site and he needs someone to write a page counter tool.  I've pasted
part of an email from the administrator at the end of this message.

When you give the administrator the page counter, be sure you also
provide a test program so that he can figure out how to use it.

And don't forget the Computer Werkerz slogan:
"Abstraction and encapsulation are the cornerstone of good code".

Thank You!
Patty


> When I create the the counter, I would like to be able to set the value it
> has.  Second, I need to be able to tell it "hit" and have it increment.
> Finally, I should be able to "reset" the counter, and "get" the current
> number of hits from it.

---
Middle Management Pooh-bah
Computer Werkerz, Inc.
pthompson@computerwerkerz.com
```



Aaron Johnson

90/100

CSCI 3415: Principles of Programming Languages
Project 4 Grade Sheet, flex and bison

Points	Possible	Description
1	5/5	Flex runs without warnings or errors
2	10/10	Bison runs without warnings or errors
3	5/5	C++ code compiles and runs
4	10/10	Monsters and treasures are optional with a Room.
5	10	A Monster’s hits and weapon may be provided in either order
6	10/10	The program reads directly from dungeon.txt
7	10/10	After parsing dungeon.txt, the user is able to either print the sorted dungeon or exit.
8	10/10	Each call to strdup() in the flex file is matched by a call to free() in the bison file
9	10/10	The code uses Monster, Treasure, Room, and Dungeon classes.
10	0/10	Rooms, Treasures, and Monsters allocated by new are released using delete in the Dungeon destructor.
11	10/10	The various strings (room description, monster type, etc.) do not have any erroneous text (e.g., leading whitespace).

==========

Excellent project, but you didn’t release your memory allocated with "new".

```
#ifndef MONSTER_H
#define MONSTER_H
#include <string>
#include <iostream>
class Monster
{
public:
        int hits;
        std::string name;
        std::string weapon;
        Monster(const std::string& n, const std::string& w, int h) : name(n), weapon(w), hits(
h) {}

        friend std::ostream& operator<<(std::ostream& out, Monster& m)
        {
                out << "<Monster>" << std::endl;
                out << "Name: " << m.name << std::endl;
                out << "Weapon: " << m.weapon << std::endl;
                out << "Hits: " << m.hits << std::endl;
                out << "</Monster>" << std::endl;
                return out;
        }
};
#endif
```

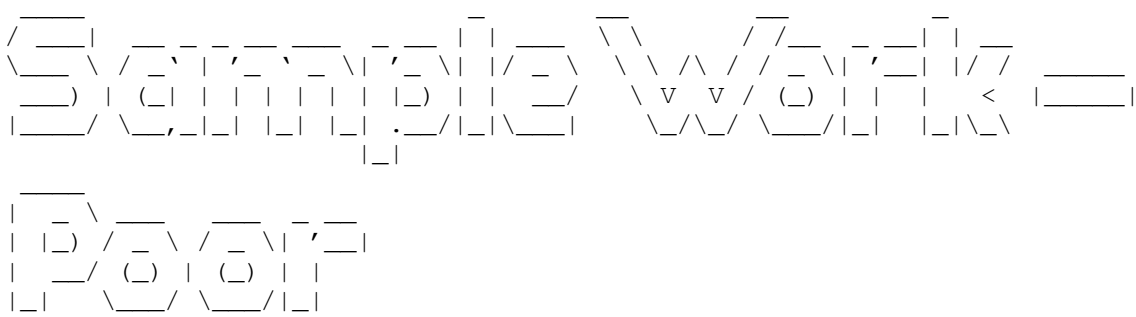
```
#ifndef ROOM_H
#define ROOM_H
#include <string>
#include <vector>
#include <iostream>
#include "Treasure.h"
#include "Monster.h"
class Room
{
public:
        int number;
        std::string description;
        std::vector<Monster*> monsters;
        std::vector<Treasure*> treasures;
        Room(int n, const std::string& d) : number(n), description(d){}

        friend std::ostream& operator<<(std::ostream& out, Room& r)
        {
                out << "<Room>" << std::endl;
                out << "Number: " << r.number << std::endl;
                out << "Description: " << r.description << std::endl;
                out << "<Monsters>" << std::endl;
                for(int i = 0; i < r.monsters.size(); ++i)
                {
                        out << *(r.monsters[i]) << std::endl;
                }
                out << "</Monsters>" << std::endl;
                out << "<Treasures>" << std::endl;
                for(int i = 0; i < r.treasures.size(); ++i)
                {
                        out << *(r.treasures[i]) << std::endl;
                }
                out << "</Treasures>" << std::endl;
                out << "</Room>" << std::endl;
                return out;
        }
};
#endif
```

```
%{
#include "Room.h"
#include "Monster.h"
#include "Treasure.h"
#include "scratch.tab.h"
%}

%%

Value:[ ][0-9]+         { yylval.myinteger=atoi(yytext + 7); return VALUE; }
Hits:[ ][0-9]+          { yylval.myinteger=atoi(yytext + 6); return HITS; }
Room:[ ][0-9]+          { yylval.myinteger=atoi(yytext + 6); return ROOM; }
Monster:[A-z0-9,' \.\-]+                { yylval.mystring=strdup(yytext + 8); return MONSTER;
}
Weapon:[A-z0-9,' \.\-]+         { yylval.mystring=strdup(yytext + 7); return WEAPON; }
Treasure:[A-z0-9,' \.\-]+               { yylval.mystring=strdup(yytext + 9); return TREASURE;
 }
Description:[A-z0-9,' \.\-]+    { yylval.mystring=strdup(yytext + 12); return DESCRIPTION; }
%%
```

```
%{
#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>
#include "Room.h"
#include "Monster.h"
#include "Treasure.h"
std::vector<Room*> rooms;
int yylex();
int yyerror(const char* s);
%}

%union {
int myinteger;
char* mystring;
Monster* monster;
Treasure* treasure;
Room* room;
void* any;
}

%token <myinteger> VALUE;
%token <myinteger> ROOM;
%token <myinteger> HITS;
%token <mystring> MONSTER;
%token <mystring> TREASURE;
%token <mystring> WEAPON;
%token <mystring> DESCRIPTION;
%type <monster> monster;
%type <treasure> treasure;
%type <room> room;
%type <any> any;

%%

any:    monster {rooms.back()->monsters.push_back($1); }
        | any monster {rooms.back()->monsters.push_back($2); }
        | treasure {rooms.back()->treasures.push_back($1); }
        | any treasure {rooms.back()->treasures.push_back($2); }
        | room {rooms.push_back($1); }
        | any room {rooms.push_back($2); }


room:   ROOM DESCRIPTION  {$$=new Room($1,$2); free($2);}

monster:        MONSTER WEAPON HITS {$$=new Monster($1,$2,$3); free($1); free($2); }
                | MONSTER HITS WEAPON {$$=new Monster($1,$3,$2); free($1); free($3); }

treasure:       TREASURE VALUE {$$=new Treasure($1,$2); free($1);}

%%

extern FILE *yyin, *yyout;

bool roomsort(Room* i, Room* j) { return i->number < j->number; }

int main(int argc, char* argv[])
{
        if(argc > 1)
                yyin = fopen(argv[1], "r");

        char input;
        std::cout << "Menu" << std::endl;
        std::cout << "(1) Display Dungeon" << std::endl;
```

```
        std::cout << "(2) Quit" << std::endl;
        std::cin >> input;
        if(input == '1')
        {
                yyparse();
                std::sort(rooms.begin(), rooms.end(), roomsort);
                for(int i = 0; i < rooms.size(); ++i)
                {
                        std::cout << *rooms[i] << std::endl;
                }
        }
        return 0;
}

int yyerror(const char* s)
{
        std::cerr << s << std::endl;
}
```

```
#ifndef TREASURE_H
#define TREASURE_H
#include <string>
class Treasure
{
public:
        int value;
        std::string type;
        Treasure(const std::string& t, int v) : type(t), value(v) {}

        friend std::ostream& operator<<(std::ostream& out, Treasure& t)
        {
                out << "<Treasure>" << std::endl;
                out << "Type: " << t.type << std::endl;
                out << "Value: " << t.value << std::endl;
                out << "</Treasure>" << std::endl;
                return out;
        }
};
#endif
```



Zachary Builter

100/100

CSCI 3415: Principles of Programming Languages
Project 4 Grade Sheet, flex and bison

Points	Possible	Description
1	5/5	Flex runs without warnings or errors
2	10/10	Bison runs without warnings or errors
3	5/5	C++ code compiles and runs
4	10/10	Monsters and treasures are optional with a Room.
5	10/10	A Monster’s hits and weapon may be provided in either order
6	10/10	The program reads directly from dungeon.txt
7	10/10	After parsing dungeon.txt, the user is able to either print the sorted dungeon or exit.
8	10/10	Each call to strdup() in the flex file is matched by a call to free() in the bison file
9	10/10	The code uses Monster, Treasure, Room, and Dungeon classes.
10	10/10	Rooms, Treasures, and Monsters allocated by new are released using delete in the Dungeon destructor.
11	10/10	The various strings (room description, monster type, etc.) do not have any erroneous text (e.g., leading whitespace).

==========

#10, I’d counsel maintaining a lists of pointers rather than of objects, then releasing them in the appropriate destructors. As written, the dynamic memory allocation isn’t doing you any benefit. That is, you’re currently just:

- allocate,
- copy (via assignment by value to vector),
- release

Under your current architecture, avoiding dynamic memory and just using objects would be correct (if not preferred). Perhaps you already know this and were just trying to appease the grade sheet? Anyway, good project.

```
//classes.h // Zachary Builter //
#ifndef CLASSES_H
#define CLASSES_H
#include <string>
#include <vector>
#include <iostream>
#include <algorithm>
class MonsterObject {
 private:
  int hits;
  std::string name;
  std::string weapon;
 public:
   MonsterObject(int h, const std::string& n, const std::string& w) : hits(h), name(n), weapon
(w) {}
   void print() {std::cout << "  Monster: " << name << "\n   Weapon: " << weapon << "\n   Hits
: " << hits << std::endl;}
};

class MonsterList {
 public:
  std::vector<MonsterObject> v;
  MonsterList() {};
};

class TreasureObject {
 private:
  int value;
  std::string text;
 public:
  TreasureObject(int x, const std::string& y) : value(x), text(y) {}
  void print() {std::cout << "  Treasure: " << text << std::endl << "   Value: " << value << s
td::endl;}
};

class TreasureList {
 public:
  std::vector<TreasureObject> v;
  TreasureList() {};
};

class MonTre {
 public:
  MonTre() {}
  void add_tre(TreasureObject& x) {t.push_back(x);}
  void add_mon(MonsterObject& y) {m.push_back(y);}
  std::vector<TreasureObject> t;
  std::vector<MonsterObject> m;
};

class Room {
 private:
   int number;
   std::string description;
   std::vector<TreasureObject> TreasureList;
   std::vector<MonsterObject> MonsterList;
 public:
   Room(int x, std::string y, std::vector<TreasureObject> t, std::vector<MonsterObject> z) : n
umber(x), description(y), TreasureList(t), MonsterList(z) {}
   Room(int x, std::string y) : number(x), description(y) {}
   void print() {std::cout << "Room: " << number << std::endl << " Description: " << descripti
on << std::endl; for(int i=0; i<TreasureList.size(); i++) {TreasureList[i].print();} for(int i
=0; i<MonsterList.size(); i++) {MonsterList[i].print();} std::cout << std::endl;}
   friend bool operator < (const Room& r1, const Room& r2) {return r1.number < r2.number;}
```

```
};
class Dungeon {
 private:
   int number_of_rooms; //not sure if I need this or not
   std::vector<Room> room_list;
 public:
   Dungeon() {number_of_rooms = 0;}
   void add_room(const Room& r) {number_of_rooms++; room_list.push_back(r);}
   void print() {std::sort(room_list.begin(), room_list.end()); for(int i=0; i<room_list.size(
); i++) {room_list[i].print();}};
};
#endif
```

```
%{
//dungeon.lex // Zachary Builter //
#include "classes.h"
#include "dungeon.tab.h"
int compute_colon_position(const char* cstar)
{
 std::string stringy = cstar;
 return stringy.find_first_of(":");
}
%}

%%

[ \t]*Value:[ ].*$ { yylval.myinteger=atoi(yytext+2+compute_colon_position(yytext)); return VA
LUE;}
[ \t]*Value:.*$ { yylval.myinteger=atoi(yytext+1+compute_colon_position(yytext)); return VALUE
;}
[ \t]*Treasure:[ ].*$ { yylval.mystring=strdup(yytext+2+compute_colon_position(yytext)); retur
n TREASURE;}
[ \t]*Treasure:.*$ { yylval.mystring=strdup(yytext+1+compute_colon_position(yytext)); return T
REASURE;}
[ \t]*Hits:[ ].*$ { yylval.myinteger=atoi(yytext+2+compute_colon_position(yytext)); return HIT
S;}
[ \t]*Hits:.*$ { yylval.myinteger=atoi(yytext+1+compute_colon_position(yytext)); return HITS;}
[ \t]*Weapon:[ ].*$ {yylval.mystring=strdup(yytext+2+compute_colon_position(yytext)); return W
EAPON;}
[ \t]*Weapon:.*$ {yylval.mystring=strdup(yytext+1+compute_colon_position(yytext)); return WEAP
ON;}
[ \t]*Monster:[ ].*$ {yylval.mystring=strdup(yytext+2+compute_colon_position(yytext)); return
MONSTER;}
[ \t]*Monster:.*$ {yylval.mystring=strdup(yytext+1+compute_colon_position(yytext)); return MON
STER;}
[ \t]*Room:[ ].*$ {yylval.myinteger=atoi(yytext+2+compute_colon_position(yytext)); return ROOM
;}
[ \t]*Room:.*$ {yylval.myinteger=atoi(yytext+1+compute_colon_position(yytext)); return ROOM;}
[ \t]*Description:[ ].*$ {yylval.mystring=strdup(yytext+2+compute_colon_position(yytext)); ret
urn DESCRIPTION;}
[ \t]*Description:.*$ {yylval.mystring=strdup(yytext+1+compute_colon_position(yytext)); return
 DESCRIPTION;}
%%
```

```
%{
#include <vector>
#include <stdio.h>
#include "classes.h"
int yylex();
int yyerror(const char* s);
Dungeon d;
%}

%union {
int myinteger;
char* mystring;
TreasureObject* tre;
MonsterObject* mon;
Room* roo;
MonTre* mat;
Dungeon* dun;
}

%token <myinteger> VALUE
%token <mystring> TREASURE
%token <myinteger> HITS
%token <mystring> WEAPON
%token <mystring> MONSTER
%token <myinteger> ROOM
%token <mystring> DESCRIPTION

%type <tre> treasureobj
%type <mon> monsterobj
%type <roo> roomobj
%type <mat> monsterandtreasure
%type <dun> dungeon

%%

dungeon: roomobj {d.add_room(*$1); delete($1);}
        | dungeon roomobj {d.add_room(*$2); delete($2);}

roomobj: ROOM DESCRIPTION monsterandtreasure {$$ = new Room($1,$2,$3->t,$3->m); free($2); dele
te($3);}
        | ROOM DESCRIPTION {$$ = new Room($1,$2); free($2);}

monsterandtreasure: monsterobj {$$ = new MonTre(); $$->add_mon(*$1); delete($1);}
        | treasureobj {$$ = new MonTre(); $$->add_tre(*$1); delete($1);}
        | monsterandtreasure monsterobj {$1->add_mon(*$2); delete($2);}
        | monsterandtreasure treasureobj {$1->add_tre(*$2); delete($2);}

monsterobj : MONSTER HITS WEAPON {$$ = new MonsterObject($2,$1,$3); free($1); free($3);}
        | MONSTER WEAPON HITS {$$ = new MonsterObject($3,$1,$2); free($1); free($2);}

treasureobj : TREASURE VALUE {$$ = new TreasureObject($2,$1); free($1);}

%%

main(int argc, char *argv[])
{
  extern FILE *yyin;
  FILE *fp=fopen(argv[1],"r");
  yyin=fp;
  yyparse();
  int exit = 0;
  while (exit == 0)
  {
    std::cout << "Menu: (type the number of menu choice and hit enter to continue)\n1) Print D
ungeon\n2) Exit\n--> ";
```

```
    int menuchoice;
    std::cin >> menuchoice;
    if (menuchoice == 1)
      d.print();
    if (menuchoice == 2)
      exit = 1;
  }
}

int yyerror(const char* s)
{
  fprintf(stderr, "error: %s\n", s);
}
```