CS 11 Data Structures and Algorithms

Course Syllabus

Skip to Main Content

jump down this page to: **schedule** | **final grades** | **assignment grades** | **exams** | **late policy** | **collaboration** | **compilers** | **getting started** | **style conventions**

Instructor

Dave Harden
Phone: (707) 527 - 4282
Email: dharden@santarosa.edu
Office Hours: Maggini Hall Room 2936, M 8-11am, WF 9-11am
Response Times: I almost always respond to emails and discussion posts within 24 weekday hours, always within 48 weekday hours. I rarely respond on weekends or evenings. My primary times for communication are MWF mornings.

Please use the class discussions for all questions whenever possible. For private questions that are not appropriate for the class discussion, email me. Do not attempt to contact me using the Canvas messaging system (or "inbox"). I don't monitor the Canvas inbox. **Please include "CS 11" in the subject when you email me!!**

Two Most Important Announcements!

- Online students **must** post an introduction in the class "introductions" discussion by Monday, January 22, or I am required to drop you from the class. When you begin working your way through the modules, you'll see that this is the first task for week 1.
- Absolutely no assignments will be accepted for any reason after Wednesday, May 16.

Textbook

"Starting Out with C++: From Control Structures through Objects", 7th edition, ISBN 978-0132576253. A more recent edition will also be fine, in case you already own one. Also, the "Early Objects" editions will probably work as well. Let me know if you need help using an alternate edition.

Schedule

The reading assignments in parentheses in the "Text Reading" column are the corresponding readings in the 8th edition of the "Objects First" edition of the text.

Assignment	Topic	Lesson	Text Reading	Suggested Start Date	Assignment Due Date
a1	Pointers	12	ch. 9 (10)	Jan 17	Jan 29
a2	Characters, Strings, Structs	13 & 14 (not provided)	ch. 10 & 11 (12.1 - 12.5 & 7.12)	Jan 29	Feb 5
a3	Operator Overloading 1	16	ch. 14.1 - 14.3, 14.5 (11.1 - 11.4, 11.6)	Feb 5	Feb 12

a4	Operator Overloading 2	16	ch. 14.1 - 14.3, 14.5 (11.1 - 11.4, 11.6)	Feb 12	Feb 19
a5	Classes with Dynamic Memory 1	17	ch. 14.4 - 14.8 (11.5 - 11.9)	Feb 19	Feb 26
a6	Classes with Dynamic Memory 2	17	ch. 14.4 - 14.8 (11.5 - 11.9)	Feb 26	March 5
a7	Inheritance	18	ch. 15 (11.10 - 11.14, 15)	Mar 5	Mar 12
midterm 1	a1 - a6	12, 13, 14, 16, 17	ch. 9, 10, 11, 14	Mar 12	Mar 12
a8	Templates/Exceptions/STL	19	ch. 16 (16)	Mar 12	Mar 26
a9	Linked Lists 1	20	ch. 17 (17)	Mar 26	Apr 2
a10	Linked Lists 2	20	ch. 17 (17)	Apr 2	Apr 9
a11	Stacks & Queues	22	ch. 18 (18)	Apr 9	Apr 16
midterm 2	a7 - a10	18 - 20	ch. 15 - 17	Apr 16	Apr 16
a12	Recursion	22	ch. 19 (14)	Apr 16	Apr 23
a13	Trees 1	23	ch. 20 (19)	Apr 23	Apr 30
a14	Trees 2	23	ch. 20 (19)	Apr 30	May 7
a15	Sorting & Searching	24	ch. 8, 19.6, 19.8 (9, 14.5, 14.6)	May 7	May 14
Final	a11 - a15	21 - 24	ch. 18 - 20	May 21	May 21

Final Grades

Your final score will be made up of the following components:

Component	**Points Each**	**Points Total**
Assignments (15)	varies	725
Discussions (15)	5	75
Midterms (2)	50	100
Final	100	100
total		**1000**

Grades will be assigned as follows: 800 for an "A", 710 for a "B", 620 for a "C", and 530 for a "D". Grades of + or - are not given.

Assignment Grades

Do not use concepts in your assignments that have not been covered in class.

Programs must compile and run correctly under standard C++ (see the section on **Compilers** below for more information). Any program submitted that does not work as specified in ways that are more than trivial will receive a score of 0. Your scores on programs will be based only on issues of style and presentation. In order to understand how to get a good score you must read the **Style Conventions** section of this document carefully. You should also pay close attention to sample solutions that are given and instructions in the lessons and text. Because of this emphasis on issues of style and presentation, students are often surprised

at their low scores on programming exercises. Be careful not to let this be you. A working program may receive a failing score!

Assignments will be scored according to the percentages in the following table. Note that the number in the first column corresponds to the number in the **Style Conventions** section, which appears later in this document.

1	Comments	20%
2	Appearance (e.g. Whitespace, Wraparound)	5%
3	Identifier Names	10%
4	Decomposition	20%
5	Indentation	10%
6	Simple Code/No Repeated Code	25%
7	Miscellaneous	10%

Exams

There will be two midterms and a final. All three rely heavily on the textbook reading assignments. You may be tested on concepts that are not covered in the lessons or assignments. All are taken online, and you can take each exam at any time of your choosing during the day on which it is scheduled (see the **schedule**). They are due at 11:59pm. Once you start, each exam must be completed within one hour, so ensure that you will not be interrupted once you begin. All exams are multiple choice. The coverage of each exam is given in the schedule. Of course, you may need to know some concepts from earlier chapters on midterm 2 and the final.

You will take the exams on the honor code. The tests are available for an entire day for your convenience, but the validity of the tests relies heavily on your academic integrity. Don't take advantage of the flexibility by sharing questions with students who have not taken the test.

You are expected to simulate a class environment when you take the exams. The exams are open book and open notes, but you cannot receive any help from another person, or a search engine, or a compiler or IDE. The rules are summarized below. Email me if you have any questions:

- Allowed: Textbook
- Allowed: Notes
- Allowed: Past assignments
- Allowed: Compiling your answers
- Not Allowed: Browsing non-class websites
- Not Allowed: Accepting assistance from another person.
- Not Allowed: Sharing questions with other students after the test.

Late Policy

This late policy is for assignments only. Late exams are not accepted.

Assignments are due at 11:59pm on the date indicated in the **schedule**. However, assignments may be submitted up to 48 hours late with no penalty. This is is the "final deadline". This does not mean that the due date is extended! For example, if your assignment is not done by the original due date, and you get severely ill between the due date and the final deadline, so that you cannot complete your assignment by the final deadline, it will be considered late. In addition, failing to complete projects by the original due date will put you behind in the class, and may delay the grading of your assignment significantly. You should make every effort to complete the assignment by the original due date.

Beyond the final deadline, assignments will be accepted until 2 weeks after the original due-date (except that no assignments will be accepted after Wednesday, May 16). They will be considered late and will receive a 50% deduction, with no exceptions. Assignments are not accepted more than 2 weeks late.

To repeat: absolutely no assignments will be accepted for any reason after Wednesday, May 16.

Collaboration

I'm letting you know up front that I am very serious about detecting and penalizing inappropriate collaboration. Please decide now that you will not engage in this. Instead get help when you need it using course discussions, email to me, tutors, etc. Last semester over 20 assignments were given grades of 0 and five students received an F in the course because of this policy.

Inappropriate collaboration first offense = ZERO on assignment and formal Academic Dishonesty Incident Report.

Inappropriate collaboration second offense = F in the course

Any variation of copying or collaborating on programming assignments, or parts of programming assignments, is prohibited. Every assignment must be 100% your own work.

You may not use even one line of code that you find online, even if you modify it. You may use websites for reference purposes (for example: how does a particular language feature work?). But you may not get information specifically related to a problem you are trying to solve (for example: what's an algorithm for reducing a fraction?).

I recommend that you get all of the information you need for the course from the text and lessons. If you need help, ask in the discussion. As often as not, information that you find online will lead you in the wrong direction anyway.

Other than participating in the course discussions, you may not work together on assignments.

If you submit code that shows similarity to another student's code or to code that is available online, you will be found in violation of this policy.

Compilers:

You will need to have a C++ compiler installed on your computer. This course requires that the assignments you turn in compile and run correctly under standard C++. I assume that you know what you need to know about compilers from your previous programming experience. Please contact me if you need help getting started with a compiler.

Course Description and Student Learning Outcomes

Continued study of computer programming including specification and implementation of data structures, and analysis of associated algorithms. Topics include: abstract data types, dynamic memory, templated functions and classes, iterators, exception handling, linked lists, stacks, queues, recursion, trees, searching, sorting, and inheritance. Several significant programming projects are written in C++.

Upon completion, students will be able to

1. 1. Use principles of software design to analyze programming problems and develop solutions.
2. 2. Create and test computer programs that incorporate complex data structures and algorithms and object oriented programming methods.

Style Conventions

[How To Get Good Grades On Your Programs]

In the real world, programmers usually work in teams and often the company that they work for has very precise rules for what kind of style to use when writing programs. For this reason, and also to encourage good programming style, we will be adopting the following style conventions for this class. This is not to say that these rules represent the only good style for writing computer programs (although in most cases they do). After you finish CS 11, you may decide that you prefer a different style than what is required here. However, in order to get good grades on your programming projects in CS 11, you must follow these guidelines.

1. **Documentation:**
   Note: If the file is a class header file or a class implementation file, paragraphs A and C do not apply.

   A. Initial File Comment: Your programs should be well-documented. Each program should begin with an initial file comment. This comment should start by indicating your name, class, date, instructor, name of file, etc. Next it should describe in detail what the program does and how the code works. Any input expected from the user and any output produced by the program should be described in detail. **You should expect your initial file comments to be at least 50 words.**

   It's good practice to comment important or potentially confusing variables at their declaration, but I won't penalize you for omitting these comments. Aside from this, in most cases it should not be necessary to place comments in the body of a function. This usually clutters up your code and ends up making the function more difficult to read. If you find yourself needing to explain something in the middle of a function, perhaps you should look for a clearer way to write it!

   B. General Advice: Your comments should be directed toward a reader who is an expert C++ programmer. You should not explain features of the language!

   C. Function Comments: Just above each of your function definitions (except main()) you must provide a comment describing what the function does. **A simple function might have a 15 word comment, while a more complex function might have a comment of at least 50 words. Make sure to explain the role of each parameter in your function comments, and refer to them by name.** Note: main() does not need a function comment, because this information should be included in the initial file comment.

   D.Comments in Classes: If this is your first time using the following guidelines for writing comments in a class, try your best to follow the instructions below, but don't worry too much about getting everything just right. You'll get full credit if it looks like you gave it your best shot.

   D.1. Header File: In the case of a class, the header file should begin with a (typically) very large initial file comment. This comment should start by indicating your name, class, date, instructor, name of file, etc. Next it should include a brief general description of the class (so client programmers can tell right away whether they want to use it), followed by a listing of all of the prototypes of public functions, each with pre and post conditions. Note that this list of prototypes is still part of the comment, so you will have to list the prototypes again in the code below this header comment. **You are required to use pre/post conditions to document your public member functions and friend functions.** Do not include any comments regarding the implementation details in the header file! This initial file comment will then be followed by the header file code (e.g. the class declaration), with no comments.

   More info about pre/post conditions: **page 1** | **page 2**

   D.2. Implementation File: In the implementation file you should start with a class invariant. (I don't expect you to have prior knowledge of what a class invariant is. The description that follows should suffice.) The class invariant will include a description of the private data members and how they are used, as well as a statement of anything that you guarantee will always be true for class objects (for example: "fraction objects will always be stored in lowest terms"). Aside from the class invariant, the

only comments you will need in your implementation file are comments on the implementation of complex functions, and comments on private functions (which do not get comments in the header file).

Here is an **outline of how this will look**.

2. **Appearance:**
   A. General: Use lots of whitespace (blank lines and spaces) to separate the different parts of your program!! Put a blank line between your declarations and your statements. Put a space before and after each operator so that instead of

   ```
   cout<<"Hello"<<x<<"my name is"<<endl<<bob;
   ```

   you write

   ```
   cout << "Hello" << x << "my name is" << endl << bob;
   ```

   Make sure your lines aren't too long, no more than 80 or 90 characters.

   B. With Functions: **Put at least 6 blank lines between function definitions.**

3. **Identifier Names:**
   A. General: Choose your identifier names very carefully. Variable names should precisely represent what the variable is storing. Do not use abbreviations unless you have seen the abbreviation used in a lesson. Don't use one letter variable names except, perhaps, in for loops.

   B. With Functions: Choose your function names so that as much as possible your program reads like English and the names describe precisely what the function does. Void function names should start with an action word (readString, getData, etc.).

4. **Decomposition**:
   Any time there is a sequence of statements in your program that performs a specific, nameable sub-task, you should consider making that sequence of statements into a function. A nice length for functions is about 10 lines, although they can be longer if they are simple (for example, lots of cout statements) or if there is just no logical way to break it up. Consider making complex functions (for example, nested loops) even shorter. A goal: when you are done with your program, I ought to be able to look at any particular function and have a general understanding of what it does and how just at a glance.

   In this class you should rarely write a function longer than 10 lines. There will be exceptions.

5. **Indentation**:
   Indents must be exactly 4 spaces.

   You may follow the indentation scheme used in the textbook or you may use the scheme used in the lectures. No others. For example, every statement must appear on a line by itself, every close curly brace must appear as the first (or only) item on a line, and every open curly brace must appear as the last (or only) item on a line.

6. **Simple Code/No Repeated Code**:
   Make sure that your code is as simple as possible and that there is no unnecessary repeated code.

7. **Miscellaneous:**
   a. In most cases no numbers other than 1 or 0 should appear in your program. Other numbers should usually be declared as global constants.

   b. **Do not use any global variables!!** Violating this guideline will cost you a lot of points!

c. You should follow the "single entry -- single exit" rule for functions and loops. This means that you should not use a statement like "break" (except in a switch statement), "return" (except in a value-returning function), "exit", or "continue".

d. Use pass by value unless you have a good reason to pass by reference. Always pass objects by reference. When passing an object by reference, use the "const" modifier when the value of the parameter should not be modified.

e. Don't mix up statements and expressions. For example, count++ should not be used as an expression, but as a statement.

f. You must use a value-returning function if (a) there is exactly one value being communicated to the calling function, and (b) there is no input or output occurring in the function.

g. Use a for loop for counter controlled loops. Do not use a for loop for any other kind of loop.

h. Don't use the fact that C++ implements `true` and `false` using the `int` values 1 and 0. For example, never use 1 or 0 in the place of `true` or `false`.

i. Use only standard C++.

j. Don't use `typedef` except to create size_type and value_type inside a class.

k. Don't use `goto`.

l. Don't use the ?: operator.

m. Don't use the preprocessor except for #include and for making sure that a header file is not multiply included (#define, #ifndef, and #endif).

n. Use initializor lists only in derived class constructors.

o. The characters "== true" or "== false" should never occur in your code.

p. You should never have

```
if (logical-expression) {
    return true;
} else {
    return false;
}
```

in your code. This can be replaced with simply

```
return logical-expression;
```

q. Don't use inline member functions