

# CS 2B Intermediate Software Design in C++

## Course Syllabus

jump down this page to: [schedule](#) | [final grades](#) | [assignment grades](#) | [late policy](#) | [collaboration](#) | [compilers](#) | [getting started](#) | [style conventions](#)

### Instructor

Dave Harden

**Email:** [dharden@santarosa.edu](mailto:dharden@santarosa.edu)

**Office Hours:** Wednesday 10:15 - 12:15, via email



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 1B" in the subject when you email me!!**

**Response Times:** I almost always respond to emails and discussion posts within 24 weekday hours, always within 48 weekday hours.

### Two Most Important Announcements!

- You **must** post an introduction in the "introductions" topic of the class forum by Monday, Apr 15, 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, June 19.

### Computer Science Department Announcements

**Opportunities for CS students** is a blog that contains announcements of internships, scholarships, free software offers, pertinent public lectures, etc. Announcements will be posted here often during the quarter.

The STEM Center, in room 4213, has CS tutors at various times each day. It is also a place on main campus where students without their own computers can do their lab work, and it also provides online tutoring for CS students. I strongly encourage you to take advantage of this valuable resource. See **the STEM Center Website** for more information.

### Official Course Outline and Student Learning Outcomes

You can access the official course outline of record for all CS courses at <http://www.foothill.edu/schedule/catalog.php>. From that page, select Dept: Computer Science -> Search, and from there, select any CS course whose official outline you want to review.

Student learning outcomes for this and other CS courses can be found [here](#)

### Textbook

Our required text will be "Absolute C++", 6th edition by Savitch. The 5th edition is also acceptable. The access code is not required. You can order this through the Foothill Bookstore at <http://books.foothill.edu/>, phone: (650) 949-7305, or you can order it online.

## Schedule

Assignments are due at 11:59pm on the date indicated in this schedule. Be sure to check the **late policy** for more information.

Assignment	Topic	Lesson	Text Reading	Suggested Start Date	Assignment Due Date
a1	Operator Overloading 1	16	ch. 8	Monday, Apr 8	Monday, Apr 15
a2	Operator Overloading 2	5.5, 15.8 - 15.9	ch. 11, 12	Monday, Apr 15	Monday, Apr 22
a3	Pointers	12	ch 10.1, 10.2	Monday, Apr 22	Monday, Apr 29
a4	Classes with DM 1	17	ch. 10.3, 9	Monday, Apr 29	Monday, May 6
a5	Classes with DM 2	17	ch. 10.3, 9	Monday, May 6	Monday, May 13
a6	Recursion	22	ch. 13	Monday, May 13	Monday, May 20
Midterm	Thru a5	12, 16, 17	ch. 8 - 12	Monday, May 20	Monday, May 20
a7	Inheritance	18	ch. 14 & 15	Monday, May 20	Monday, May 27
a8	STL, Templates	19	ch. 16, 18, 7.3	Monday, May 27	Monday, June 3
a9	Linked Lists 1	20	17.1 - 17.3	Monday, June 3	Monday, June 10
a10	Linked Lists 2	20	17.1 - 17.3	Monday, June 10	Monday, June 17
final		18, 19, 20, 22	ch. 14, 16, 17, 18, 22	Monday, June 24	Monday, June 24

## Final Grades

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

Component	Points Each	Points Total
Assignments (10)	85	850
Discussions (10)	5	50
Midterm	50	50
Final	50	50
<b>total</b>		<b>1000</b>



Grades will be assigned as follows: 900 for an "A", 800 for a "B", 700 for a "C", and 600 for a "D". Grades of + or - are rare and at the instructor's discretion.

## Assignment Grades

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 that use concepts that have not been covered in the class may receive a score of 0.

Programs will be scored according to the percentages in the following table. 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)	10%
3	Identifier Names	10%
4	Decomposition	20%
5	Indentation	10%
6	Simple Code/No Repeated Code	20%
7	Miscellaneous	10%



## Exams

There will be a midterm and a final. Both rely heavily on the textbook reading assignments. You may be tested on concepts that are not covered in the lessons or assignments. Both exams are taken online, and you can take each exam at any time during the day on which it is scheduled (see the **schedule**). Once you start, each exam must be completed within one hour, without exception, so make sure to ensure that you will not be interrupted once you begin. Both exams are multiple choice. The midterm covers the topics in assignments 1 through 5. The final emphasizes the topics in assignments 6 through 10, but may also touch on topics from the earlier assignments.

You will take the exams on the honor code. The tests are available for one full 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, and you may even use your compiler, but you cannot receive any help from another person. 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**. Assignments must be turned in by the date listed in the course schedule. However, assignments may be submitted up to 48 hours late with no penalty. This 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, June 19). 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, June 19.

## 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. Every semester many assignments are given grades of 0 and a few students receive 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?). Don't ask for help from online forums. They will almost invariably do the problem for you, or give you bad information.

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 an SRJC student you can also use NetTutor for free. 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.**

Examples of specific actions that violate this policy:

- Using chegg.com or coursehero.com for any reason
- Using an online tutor
- Viewing another student's code
- Allowing another student to view your code
- Viewing or copying code online that is specifically related to a course assignment

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 with any ANSI/ISO standard C++ compiler. You are free to use any C++ compiler you choose; however, I recommend Visual Studio if you are working in Windows and XCode if you are working on a Mac. There are tutorials in the Week 1 Module in Canvas to help you get started with these.

## 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 2B, 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 2B, you must follow these guidelines.

## 1. Documentation:

**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.

Important local variables should be commented at their declaration. 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 you must provide a comment describing what the function does. **A simple function might have a 15 word comment, while a more complex function should 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.**

**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 header comment. This comment should include a general description of the class (so a client programmer can tell right away whether she wants to use it), followed by a listing of all of the prototypes of public members, each with pre and post conditions. Note that this list of prototypes is still part of the comment. 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.** Do not include any comments regarding the implementation details in the header file! This very large 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!! When I look at your program my first impression should not be a page crammed with code. Get rid of wraparound. 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 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 feel the identifier name would otherwise be so long that it would hinder the readability of your program. 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 CS 2B you should rarely write a function longer than 10 lines.**

#### **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. Use the "const" modifier when you are passing a parameter by reference but the value of the parameter should not be modified.
- e. Always treat statements and expressions as distinct entities. For example, count++ should not be used as an expression, but as a statement. This will usually result in clearer, simpler code (although it may result in more typing).
- 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. Never 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. Never use `goto`.
- l. Never 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 initializer lists only in the context of inheritance.
- o. The characters `"== true"` or `"== false"` should never occur in your code.
- p. You should never have

```
if (x) {  
    return true;  
} else {  
    return false;  
}
```

or

```
if (x) {  
    return true;  
}  
return false;
```

in your code (where `x` is any expression). These can be replaced with simply

```
return x;
```

- q. Don't define member functions inside a class declaration.