Mid-Term Review

The purpose of this document is to help you prepare for the midterm exam in CSC 100. This is something that you should do for any test you take in any class: Go through the material that is covered, and outline a list of topics that have been covered. It is generally your responsibility to do this, and usually it's not something instructors will provide for you. Regardless of whether you are given a list of topics, it's a good exercise and memory refresher to scan through the original materials (your notes, outside of class readings, any provided lecture slides/materials, lab write-ups, etc.) and summarize what has been covered. A good exercise for this class would be to set this handout aside, go through the material yourself and make a list, and then compare with my list. This will help you see if you have a good picture of the topics that are important - is anything missing from your list?

The midterm consist primarily of short answer questions, where short answer questions can involve BYOB programming topics, and there will be some terminology questions as well. You might be asked to sketch out a BYOB script to solve a particular problem, or to describe what a provided BYOB script does. If you are asked to sketch out a script you should use the "writing style" described at the end of this document - it would be worthwhile to practice writing things out like this before you come to the exam! The main purpose is to clearly express your understanding of algorithmic solutions, not to demonstrate any memorization of BYOB blocks. For example, if you are sketching out a script that inserts an item in a list, you would write that out something like "insert (player) at 1 of (player list)". What's not very important is the precise wording (did you use "at" or "in", etc.). What *is* very important is that you demonstrate something that makes sense as far as the underlying concept of inserting something in a list. Inserting something in a list must include three parameters: what you are inserting in the list, where you are inserting it in the list, and what list you are inserting into. It simply doesn't make sense to talk about inserting something into a list unless you give those three pieces of information - the precise wording of how you express that is much less important than the concept that it represents, and how that reflects your mental model of what an insertion into a list does.

The rest of this document is broken down into the three main sources of content in this class: The lectures, *Blown to Bits* readings, and lab exercises. Once you have a list of topics (my list or your list), you should go through it carefully and think: "Do I remember what was covered on this topic, and do I understand it?" "What kinds of questions might be asked about this topic, and do I know how to answer them?" Even better would be a study group where you go through the list of topics and some people in the group make up questions for topics, and the others propose answers (and then rotate roles in the group).

Lecture Topics

While all topics are important (remember that!), one particularly useful strategy would be to look through the lecture slides and find the places where specific questions were posed to the class. If you were asked to solve some problem in class, then that is an indication that it might be a good test question!

Lecture 1: Class Introduction

- Note: None of this is content for the exam

Lecture 2: What is Computer Science

- Basic definition and history
- Note: We did not cover past slide 14, and you aren't responsible for this (some were added to a later lecture, though)

Lecture 3: Abstraction

- General principles: detail removal and generalization
- Using BYOB blocks (or functions and procedures) for abstraction

Lecture 4: Data Representation - Part 1, Numbers

- Number representations and bases
- Bases: Decimal, binary, and hexadecimal
- Base conversions:
    - Between binary and decimal
    - Between hexadecimal and decimal
    - Between hexadecimal and binary

Lecture 5: Algorithms - Part 1, The Basics

- Definitions and concepts
- Problems vs algorithms
- Algorithm properties and trade-offs
- Computing speed and data sizes inspiring algorithms work

Lecture 6: Algorithms - Part 2, Time Complexity

- Meaning of "time" for algorithms
- Constant time
- Basic loop patterns and time complexity
- Linear time
- Quadratic time
- Logarithmic time (and binary search)
- Exponential time

Lecture 7: Concurrency

- Single core vs multi core models
- Types of concurrency
- Parallel algorithms and speedups
- Race conditions
- Deadlocks

Blown to Bits Readings

The Blown to Bits readings cover just a few core topics of how technology is affecting the world and our daily lives, and the impact of that on personal privacy and other issues. The core topics are illustrated with a lot of individual stories. While the individual stories are not so important for the factual details they contain, they are important as good illustrations of the core topics, so make sure you know the basics of these stories. You might be asked, for example, to describe an instance in which information is embedded in a digital file that reveals more than what is obvious to a casual observer (and you could then describe one of the following stories: incorrect redaction of the Italian journalist shooting, Word documents published with "track changes" enabled, cameras storing the camera serial number in image files, …).

Blown to Bits: Chapters 1-3 (Only the high points are listed here)

- Prevalence (ubiquity!) of digital data, and information represented in digital form (bits)
- Exponential growth of computing power and data capacity over time
- Digital "footprints" and "fingerprints"
- RFID tags and other ways of tracking people or things
- File formats and information hidden in digital files
- Steganography: intentionally hiding data in a file
- File deletion: what really happens when a file is deleted

Lab Exercises

The labs give you experience with BYOB programming, but also introduce some big concepts that are important well beyond just getting the lab done. While you need to be familiar with BYOB programming, and you need to be able to sketch out programs and describe how BYOB scripts work, the most important thing with the labs is to have a strong understanding of the big concepts and how they apply not only in that particular lab but in general. The other tip with the labs is to go through the lab quizzes - these are designed to ask questions about the most important concepts in the labs. If you understand the answers to these questions (really understand why the answer is what it is, not just that you can repeat the answer), then you understand a lot of the main points in the labs. Those quizzes aren't a comprehensive coverage of important topics, of course, but they're a good start! You should also go through the list of terminology that is in each lab write-up - this covers the basic terminology we use, so you should be comfortable with all of these words and phrases!

Lab 1: Introduction to Scratch/BYOB - animations and communication

- BYOB concepts - sprites, scripts, costumes, etc.
- Blocks, parameters, and arguments
- Events and broadcast messages
- Big concepts: Problem decomposition, event-driven programming, concurrency, and versioning

Lab 2: Making things interesting with interaction, variables, and more

- Sprite movement and smooth animation
- Use of variables in BYOB scripts
- Modeling physical objects
- Software testing
- Solution elegance

Lab 3: Abstraction with Functions

- Types of BYOB blocks: command (which can be regular or C-blocks), hat, and reporter (which can be a predicate or a general reporter)
- Important terminology: function, procedure, parameters, arguments, reporting, and returning
- Input checking and robustness
- Boolean operators and truth tables

Lab 4: Starting a simple math tutor program - and more interaction

- Scope of variables (global, sprite local, or script local)
- Reductions
- Top-down vs. bottom-up design

Lab 5: Using lists for data

- Basic list concepts and operations: retrieving items, inserting/adding items, deleting items, replacing items, and checking whether a list contains an item
- How to experiment with BYOB to determine behavior of different operations
- Swapping items in a list
  "Move to front" idea and why it is useful

Lab 6: Experimenting with algorithms

- Searching and sorting in lists
- Linear vs quadratic time
- Timing scripts
- Instrumenting code and counting operations

Writing BYOB scripts on paper

Expressing and defining algorithms is a vital topic for this class. It absolutely pervades every area of computer science, and you need to be able to describe algorithms in a clear and unambiguous way - on paper as well as when you are working on a computer. The best language we have in this class, and the one that you have gained experience with, is BYOB. Unfortunately, as a graphical programming language, based around dragging out graphical puzzle pieces, it doesn't lend itself very well to writing on paper. Because of that, we have developed the following technique for sketching out BYOB code on paper.

When writing on paper, we will use simple visual annotations to mark the important parts of BYOB scripts and blocks. In particular, the important visual elements will be represented as shown below, where examples have the "hand drawn" version next to each BYOB example.

- Hat blocks: Designate by surrounding the event that triggers the hat block with lines. For example:

when green flag clicked

- Variables: When writing a variable, surround it with parentheses, so initializing a variable sCount to 0 would look like this:

set (sCount) to 0

- Block arguments: When you use a block, there are two distinct things that are shown: descriptive title words, and arguments/parameters. On the computer, the parameter slots show up as little "windows" that you can fill in. On paper, what we will do is underline parameters, like this:

go to x: 0 y: 0

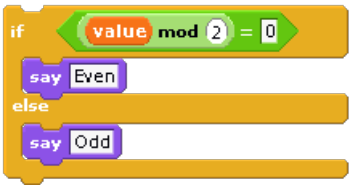
Note that sometimes arguments can include blocks that also take parameters - in that case things would be underlined twice (or more!), as in the following example (note also the use of a variable):

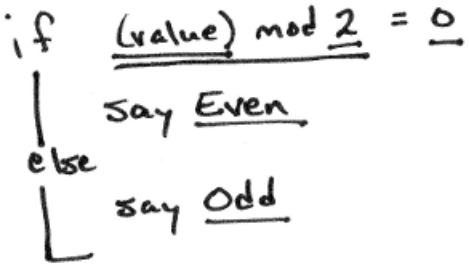
Say join Hello (name) for 2 secs

- C-blocks: for any block that contains other blocks, draw a vertical line down from the top of the C-block that marks the blocks that it contains (and the blocks that are contained will be indented slightly to the right because of this):

repeat 8
move 10 steps

The if/else block has two “container” areas, so is a special case - it is drawn as follows:





Finally, as a bigger example, this is how we would write out a block definition for a block that adds up the values in a list (this is the “accumulator pattern” example from Lab 5):