COP 3502: Computer Science I
Spring 2004

-Day 2 – January 7, 2004 –
Introduction to Algorithms

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CC1 211, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2004

School of Electrical Engineering and Computer Science
University of Central Florida

What is an Algorithm?

- Computers are devices that do only one kind of thing: They carry out *algorithms* to process information.
- To computer scientists, the algorithm is the central unifying concept of computing, the mode of thought that is the core of the computing perspective.
- What is an algorithm?
  - A set of logical steps to accomplish a task.
  - A "recipe of action".
  - A way of describing *behavior.*

Everyday Algorithms

<u>Recipe for Chocolate Chip Cookies</u>

Ingredients:
2 ¼ cups flour1 tsp salt
1 tsp baking soda2 eggs
¾ cup brown sugar1 tsp vanilla extract
¾ cup granulated sugar1 cup soft butter
12 oz. semi-sweet chocolate chips

Steps:
Preheat oven to 375 degrees
Combine flour, salt, baking soda, in bowl. Set mixture aside.
Combine sugars, butter, vanilla and beat until creamy.
Add eggs and beat.
Add dry mixture and mix well.
Stir in chocolate chips.
Drop mixture by teaspoons onto un-greased cookie sheet.
Bake 8 to 10 minutes

Bad Algorithms

- What is wrong with the following algorithm? (From the back of a shampoo bottle.)
  - Directions: Wet hair. Apply a small amount of shampoo, lather, rinse, repeat.

Answer: It never ends!

Computer Algorithms

- In the realm of computer algorithms, an algorithm is useful only if:
  - The algorithm accepts input data (not all do, however).
  - The algorithm processes that data in some fashion.
  - The algorithm produces some output (the results).

Computer Algorithms (cont.)

- However, to be a correct algorithm, it must correctly solve the problem for any valid input data.
  - Also, for the same input data, it must always give the same answer.
  - Invalid input data should produce an error message or some other indication that the algorithm cannot correctly solve the problem. It should not produce an answer when given incorrect data since the user will think that the answer is valid.

Computer Algorithms (cont.)

- Successful algorithms must consider all possible cases presented by acceptable data. You will succeed more quickly at constructing algorithms if you make it a habit to:
  - Think about the problem and its data.
  - Enumerate all the special cases that the algorithm must handle.

Describing Algorithms

- In specifying behavior, an algorithm must be:
  - Precise
  - Unambiguous
  - Complete
  - Correct
- There are various techniques that can be used to describe algorithms:
  - Natural language (English)
  - Pictures (flow-charts)
  - Pseudocode or a specific programming language

A Natural Language Algorithm

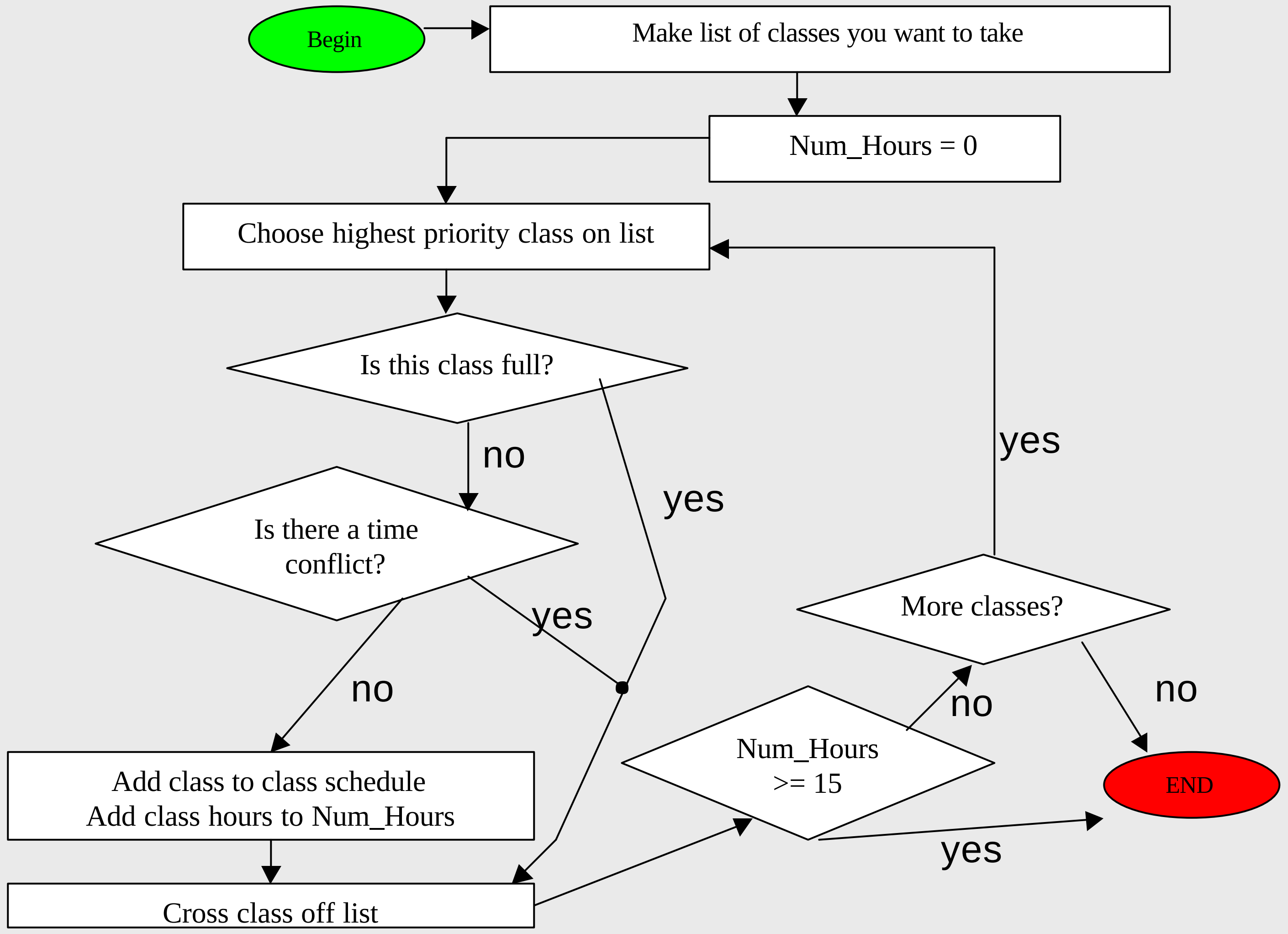
- Consider an algorithm for registering for classes.

1. Make a list of courses you want to register for, in order of priority.
2. Start with an empty schedule. Number of hours = 0.
3. Choose highest priority class on list.
4. If the chosen class is not full and its class time does not conflict with any class already in the schedule, then register for the class:

   4a. Add the class to the schedule.

   4b. Add the class hours to the number of hours scheduled.
5. Cross that class off your list.
6. Repeat steps 3 through 5 until the number of hours scheduled is >= 15, or until all classes have been crossed out.
7. Stop.

Flowchart Representation


Begin
Make list of classes you want to take
Num_Hours = 0
Choose highest priority class on list
Is this class full?
no
yes
Is there a time conflict?
yes
no
Add class to class schedule
Add class hours to Num_Hours
Cross class off list
Num_Hours >= 15
no
yes
More classes?
yes
no
END

Properties of Good Algorithms

1. Precision
   - Each step must be clear and unambiguous in its meaning.
   - The order of execution of the steps must be clear.
   - The number of steps must be finite. Each step must be finite.
2. Simplicity
   - Each step must be simple enough that it can be easily understood.
   - Each step should translate into only a few (or one) computer operation(s) or instruction (s).
3. Levels of Abstraction
   - The steps in the algorithm should be grouped into related modules or blocks.
   - Modules may be nested (one inside another).
   - Other algorithms may be referred to by name rather than including all of their steps in another algorithm.

Algorithms and Abstraction

- *Abstraction* refers to the logical grouping of concepts or objects. This allows you to define and implement in general terms without requiring or specifying the details.
- Well-defined algorithms are organized in terms of abstraction. This means that we can refer to each of the major logical steps without being distracted by the details that make up each one.
- The simple instructions that make up each logical step are hidden inside *modules.* These modules allow us to function at a higher level, to hide the details of each step inside a module, and then refer to that module by name whenever we need to use it.

Algorithms and Abstraction (cont.)

- Modularization allows us to:
  - Build and test each module independently.
  - Interchange equivalent modules.
  - Reuse modules whenever/wherever required.
- By hiding the details inside appropriate modules, we can understand the main ideas without being distracted. This is a key goal of using various levels of abstraction.

Algorithms and Abstraction (cont.)

- Each module represents an abstraction. The name of the module describes the idea that the module implements. The instructions hidden within the module specify how that abstraction is to implemented.
- We can see what is being done (the idea) by reading the descriptive name of the module without having to pay attention to how it is being implemented.
- If we want to understand how it is implemented, then we can look inside the module to find out.

Software Lifecycle and Algorithm Development

1. Understand the problem
    - The problem must be completely understood in order to determine what is required for its solution.
    - In the university/learning environment this means that you need to read the problem carefully!

2. Analysis
    - Identify the problem inputs and outputs.

Software Lifecycle and Algorithm Development (cont.)

3. Design
   - Develop a list of steps (algorithm) to solve the problem.
   - Refine the steps of this algorithm. (divide and conquer).
   - Verify that the algorithm solves the problem (correctness).

4. Implementation
   - Implement the algorithm as a program (C in this course).
   - Must know the specific language used for implementation.
   - Convert steps of the algorithm into programming language statements.

Software Lifecycle and Algorithm Development (cont.)

5. Testing and Verification
   - Test the complete program (modules independently) and verify that it works as expected.
   - Use different test cases (not just one) including critical test cases.
6. Maintenance
   - Long term maintenance and support for the algorithm and software.

Algorithm Refinement

- Consider the following algorithm for drinking a glass of water.

  1. Enter the kitchen.
- 2. Get a glass.
- 3. Get the water from the refrigerator.
- 4. Fill the glass with water.
- 5. Drink it.

Algorithm Refinement (cont.)

- Refinement of step 1.

  1.1 Walk to the kitchen door.

  1.2 Open the door.

  1.3 Walk into the kitchen.

- Refinement of step 3.

  3.1 Open the refrigerator.

  3.2 Get the water.

  3.3 Close the refrigerator.

- Refinement of step 4.

```
4.1  While the glass is not full
     4.1.1  Pour some water into the glass
```