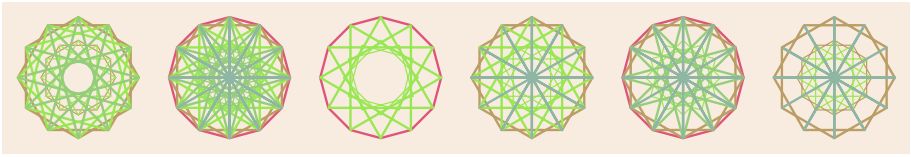
Review

Professor Leon Tabak

31 May 2022



Our programming language

- C programming language
  - 50 years old (developed with UNIX operating system)
  - influential design—basis for many languages that have followed
  - a compiled language
  - "structured assembly language"
    - declare variables with types (**int**, **double**, and others)
    - use white space, comments, and descriptive names of variables and functions to make it easy for readers to understand source code
    - choose among alternative actions with **if** and **switch** statements
    - repeat actions with **for** and **while** loops
    - create data structures with arrays and **struct**s
    - make source code easier to understand by dividing large program into smaller functions

    - compile functions separately, then link object files
- pointers
    - pointers = addresses
    - programmer allocates a block of memory
    - calloc () (or malloc()) function returns address of allocated block of memory
    - programmer frees memory that is no longer needed (with free () function)
    - programmer can add to/subtract from pointers
    - this can produce addresses of memory programmer should not try to read or write!
- prepare program for execution in 4 steps
    - preprocessor (**#define** and **#include**)
    - compiler—check grammar, translate to assembly language
    - assembler—translate to machine code
    - linker—combine object code from several sources, build executable file
- a systems programming language
    - compilers
    - operating systems
    - device drivers
    - robots, drones, embedded applications
    - networking / data communications
- assessment: what do we think of C?
    - powerful!
    - dangerous!
- what is missing?
    - garbage collection
    - object-oriented programming
    - functional programming

Questions about our programming language

1. Who created the C programming language?
2. Where did the creator of the C programming language work?
3. The creator of the C programming language won the Turing Award. Search online with the name of this person and the words "Turing Award."

   Read the citation.

   Share something interesting that you learned from reading the citation.

4. Name two programming languages whose design was strongly influenced by the design of the C programming language.

5. What is the difference between **int** and **int***?

6. Write code the prints the first 12 positive odd integers.

7. Write code that creates an array of integers and fills it with the first 8 prime numbers.

8. Write code that creates a fraction, assigns 2 to the numerator, and assigns 3 to the denominator.

```
struct fraction {
    int numerator;
    int denominator;
};
```

Software engineering in a first course in computer science

- expect challenges!
  - writing a program can take a long time
  - even the most talented and experienced programmers encounter frustrations
- write your code for 2 audiences
  - the computer
  - your teammates, your future self, and other people
- divide a big problem into smaller problems
  - avoid repeating code
  - write functions
  - keep functions short
    1. say what the function will do in your own language—if you cannot say it in Chinese or English, you cannot say it in C!
    2. give function a meaningful name
    3. decide what information the function needs to do its job—specify the number and types of parameters
    4. decide what kind of information the function will return to its caller—specify the return type

5. specify the sequence of logical and arithmetic operations that will produce the result you want

- use stub functions
  - write stub function by completing the first 4 steps of the recipe for writing functions, then delaying step 5 until you know more
  - a stub function will compile and run
  - a stub function returns the same result everytime
  - a stub function will almost always return an incorrect result
  - a programmer can continue working on other parts of the program
  - here is an example...

```
double squareRoot( double x ) {

    // I do not know how to compute a square root,
    // so I will say that square root of every number
    // is 1.0!

    // Later, I will find an algorithm for computing
    // square roots. Then I will add code to this function.

    // By writing a stub function now, I am giving myself
    // the ability to continue to work on other parts of
    // my program until I find a teammate, book, or online
    // article that will help me finish writing this
    // square root function.

    return 1.0;
} // squareRoot( double )
```

- work with other people
  - build on the work of others
  - invite others to look at your code
  - show your work in progress
  - return to your clients again and again to make sure that you are producing a program that will solve their problem
- build a program in small steps
  - write a few lines of code
  - compile
  - test/ run the program
  - locate and fix any errors
  - repeat

Questions about software engineering

1. True or false: With enough study and practice, we will eventually learn how to write computer programs easily, quickly, and without errors.
2. What is an advantage of compiling and testing a program frequently?

   Why might a choice to write lots of code before compiling and running the program again make our work harder?
3. What is the first step in writing a function?
4. It is not enough to write code that compiles and runs. We have to write code not just for the computer, but also for another audience. Which other audience?
5. What is a use of stub functions?

Algorithms in a first course in computer science

- study searching and sorting algorithms
  - because many programs that we will write will search and sort
  - because these algorithms are simple enough for beginnning students—simple to write, simple to understand
  - because experts have studied these algorithms thoroughly—we know what the characteristics of these algorithms are
  - because a study of these algorithms will prepare us to write and analyze other algorithms later
- searching and sorting are related
  - searching in a list is easier if we sort the list first
  - to sort a list, we must search in the list
- 2 search algorithms
  - sequential search
    - useful when searching in a list that has not been sorted
    - must examine every element of list
    - $O(N)$ (linear time)
    - written with a **for** loop
  - binary search
    - useful when search in a list that has been sorted
    - the way that we search in a dictionary or telephone book

        * "divide and conquer"—repeatedly divide list in half
        * do not have to touch every element of list
        * $O(\log N)$ (logarithmic time)
        * can be written recursively (function calls itself) or iteratively (with a **while** loop)

- 3 sort algorithms
    - selection sort
        * $O(N^2)$
        * repeatedly search for the smallest element in the still unsorted part of the list (the tail end of the list)
        * N steps
            · at step k, first k elements of list are in order
            · at step k, first k elements are the smallest k elements in list
            · at step k, search through $N - k$ elements
            · search from left to right in tail end of list
            · search goes all the way to the end of the list every time
        * **for** loop
    - insertion sort
        * $O(N^2)$
        * ... but is on average faster than selection sort
        * best case: list is already in order
        * best case: $O(N)$
        * N steps
            · at step k, first k elements of list are in order
            · at step k, first k elements of list are the same k elements that were at the beginning of the list before sorting began
            · at step k search in the sorted part of the list for the place in which to insert the k^{th} element
            · search is from right to left
            · search goes until it finds the right position
            · search will not always go all of the way to the beginning of the list
        * **while** loop
    - merge sort
        * $O(N \log N)$
        * "bottom up" and "top down" versions of algorithm
            · "bottom up"—iterative (a loop)
            · "top down"—recursive (function alls itself)
        * key step—merge two sorted lists to make a bigger sorted list

- 3 **while** loops
- requires enough memory to hold two copies of list
- fewer comparisons than selection or insertion sorts, but more moving data

Questions about algorithms

1. Why do we study searching and sorting algorithms?

   Answer:

   (a) Study of these simple and well-understood algorithms teaches us how to design and analyze other algorithms.

   (b) Many of the problems that software engineers are called upon to solve require some searching and sorting.

2. Which of the two searching algorithms that we studied (sequential search and binary search) is faster?

3. What property must a list have before we can search in the list using the binary search algorithm?

4. Which of the three sorting algorithms that we studied (selection sort, insertion sort, and merge sort) is fastest?

5. The performance of one of the sorting algorithms depends on how well ordered the list is before sorting begins.

   Is it the selection sort or insertion sort?

6. Two of the five algorithms that we studied (sequential search, binary search, selection sort, insertion sort, merge sort) invite recursion.

   In the recursive versions of these algorithms, a function does something with a list, divides the list in half, and then calls itself to do the same thing with a smaller list. The process stops when the division produces a list that contains only one element.

   Which of the two algorithms might we naturally describe with recursion?

What lies ahead

- Learn how to analyze algorithms. Learn how to show that an algorithm is $O(N^2)$ or $O(N \log N)$.
- Study more sorting algorithms.

  - quicksort
  - heapsort

- Learn how to construct and use other data structures.
  - linked lists
  - binary search trees
  - balanced trees
  - hash tables
  - stacks and queues
  - priority queues

- Explore use of algorithms data structures in particular domains.
  - operating systems (creating processes, communication between processes, scheduling, management of memory)
  - distributed computing
  - data bases
  - computer graphics
  - . . . and so on . . .