Genome 540 Discussion

Conor Camplisson

January 3\textsuperscript{rd}, 2023
About Me

Background
• from San Diego, CA
• Undergrad: biochemistry
• Gap years:
  • academia: in situ sequencing dev
  • industry: ReadCoor, Inc. startup
• UWGS: 4\textsuperscript{th} year with Brian Beliveau

Wet lab
• In situ “omics” tech dev
• Multiplexed FISH tech dev

Dry lab
• Design / analysis pipelines
• Specific probe design
• DNA library design

For fun
• Programming, web dev
• Sailing, skiing
Choosing a programming language

Python
- “interpreted”
- dynamic-typed
- data type is not required while declaring variable

C/C++
- “compiled”
- statically typed
- data type is required while declaring variable
- debugging is challenging

A = 5
B = “x”
C = [‘x’, ‘y’, ‘z’]

int A = 5;
char B = ‘x’;
Utilizing existing helper functions in C++

**function**

**qsort**

```cpp
void qsort (void* base, size_t num, size_t size,
            int (*compar)(const void*,const void*));
```

**Sort elements of array**

Sorts the `num` elements of the array pointed to by `base`, each element `size` bytes long, using the `compar` function to determine the order.

The sorting algorithm used by this function compares pairs of elements by calling `compar` pointers to them as argument.

The function does not return any value, but modifies the content of the array elements as defined by `compar`.

The order of equivalent elements is undefined.

[Code example]

```cpp
int values[] = { 40, 10, 100, 90, 20, 25 };
int compare (const void * a, const void * b)
{  
    return ( *(int*)a - *(int*)b );
}
int main ()
{
    int n;
    qsort (values, 6, sizeof(int), compare);
    for (n=0; n<6; n++)
        printf ("%d",values[n]);
    return 0;
}
```

Programming Style

• The more readable your code is
  • The more I’ll be able to help you if something’s wrong
  • The more useful it will be to you later
  • Try to use Github (version control and easy sharing)

• Tips for readability
  • Intuitive variable/function names, (a, b, c vs. seq_name, seq_length)
  • Comments
    • Outlining general structure of program/key points of implemented algorithm
    • Clarifying any tricky/unintuitive lines of code
  • Simplicity over performance optimization (until it becomes necessary)
  • Write more functions
  • Write less “for” loops
General tips for homework

- C/C++ or Java for some homework are necessary
- Python works for most homework
- R and MATLAB are not suitable

- It is important to understand biological questions
- “Gap” between understanding the algorithm and writing it out
- Your program might work on the test data but not work on the real data (much bigger and some unexpected issues)

- Print intermediate output
- It’s not necessary to write every basic function (for example, sort)
- Please make an effort to match the template!
General tips for homework

• Start with pseudocode

• Start writing and then improve the details

• C++ is much faster than Python (at least for HW1)

• Get comfortable with pointers

• Output to template directly
  • Programmatically generate homework output to turn in
  • Format will be checked programmatically
General tips for homework

My personal experience of each pset:

• [study the spec] What (exactly) are the high level goal(s)?

• [brainstorm] In general, how would I achieve those goals?
  • [review] (do I understand the algorithm/concepts from lectures?)

• [brainstorm] What is a reasonable first step to accomplish?
  • [brainstorm] How do I do that step in <language>?

• [dev] implement the step

• [brainstorm] how can I be sure that I did this step correctly?
  • [brainstorm] what would be a good test to pick up on this?

• [dev] implement some basic testing (e.g. print statements)

• [dev] test the code

• This step works, start thinking of the next step

• [...]
Programming tips - testing

• Create small, easily-verified test cases

• Edge cases

• Print intermediate output

• Write incrementally, test as you go
  • Assertion statements

• Check against expectations
Programming tips - efficiency

• Remove unnecessary stuff from loops

• Avoid slow comparison routines when sorting

• Profiling tools
  • line_profiler (python)
  • gprof, valgrind (C/C++) [valgrind also identifies memory leaks]
  • dprofpp (Perl)
Topics for future discussion sections?

• Scalable and reproducible bioinformatics pipelines (Snakemake)
• Parallel computing: threading, multiprocessing, cluster computing
• General programming tips, languages (Python, C++, Unix tools)
• Version Control / Github
• Jupyter Notebooks / JupyterLab
• Recent DNA sequence algorithms from literature
• Virtual environments: how and why
• Relational database design