Programming in Engineering
PiE CTW-MSM 191158510

Exam C/C++ (2012)

Instructions

Present your results in a report that includes a short explanation of your approach, answer to the questions, what you did and a description of each plot and output file. Please do not include full-source code, if required you can use code segments to illustrate your point. You can come to the staff for help. Hand in your report before August 6th, by e-mail to `vogarko@gmail.com`. The report must be submitted as one file; a pdf and all plots should contain axis labels and detailed captions. Furthermore, please, submit the C/C++ files that you create along with the report in one compressed archive. Submit all the C++ files (no executables) you needed so that we can compile and run the code ourselves (state in the report which platform you developed your code on, for example Linux, Windows, Mac, ...). There is no minimum or maximum length, but all parts of the questions should be answered.

Provide your name and student number in all files and documents you send to us.

Grading: Your grade will be determined by the quality of your report, code (including comments `// we advise to write comments in English`) and your performance in the oral exam. In order to pass the class, you should have solved all of the problems on the examination sheet and be able to explain your work during the oral exam. To obtain a grade eight or higher, the code should be efficient, in good style and very well commented. For the top grades (9 and 10 points), the starred harder question should be undertaken.

Oral Exam: After having received your submission, we will check it and invite you for an oral exam. For this please bring a printout of the report and your laptop with the source files, so that we can ask you about the code. The exam will be administered by Vitaliy Ogarko in Horstring Z121.

To avoid typical problems:

- Do not include executables in the attachment. It may cause problems with sending emails. Use .zip archives, not .rar.
- Use dynamic arrays (i.e., *int *a = new int[1e5];*), and better avoid using valarray.
- Use markers for plotting data points and lines for extrapolation.
- Learn how typical functions (power, exponential) look in the log-log plot.
- Use figures instead of tables for presenting the data; choose color schemes that look good in black and white.

Exercise 1.1

This part of the exam requires that you only answer in words (no programming).
Before you copy this file into your computer and compile it - read it and write down in maximal two sentences what it does.

```
#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;

int main(int argc, char *argv[])
{
   const int ifmax = 15;
   int fi[ifmax] = {5, 2, -1, 2, -2, 5, 5, 1, -4, -3, 3, 5, 1, -4, -1};
   int ftmp = 0;

   // write original field
   for (int i=0; i<ifmax; i++)
      cout << fi[i] << " ";
   cout << endl;

   for (int i=0; i<ifmax; i++)
   {
      for (int j=ifmax-1; j>=1; j--)
      {
          if (fi[j] < fi[j-1])
          {
             ftmp = fi[j-1];
             fi[j-1] = fi[j];
             fi[j] = ftmp;
          }
      }
      // control output if needed
      for (int ii=0; ii<ifmax; ii++)
         cout << fi[ii] << " ";
      cout << endl;
   }

   // write final field
   for (int i=0; i<ifmax; i++)
      cout << fi[i] << " ";
   cout << endl;

   return 0;
}
```

Exercise 1.2

1. Given N numbers, how many operations are required in this algorithm?

2. Rewrite this code without the line "using namespace std;" and using dynamic array instead of static.

Exercise 2

A prime number is defined as 'a positive integer $p > 1$ that has no positive integer divisors other than 1 and p itself.'

The brute force, but simple, method to decide if p is prime is to test if (n mod $q \neq 0$) for all numbers $1 < q < p$. A quicker way to find primes is to only perform the division test for *prime numbers* less than p. For example, to test the number 11 you only need to check if (11 mod 2), (11 mod 3), (11 mod 5) and (11 mod 7) are zero.

1. Write a C++ program to compute the first N prime numbers, where N is given by the user. Use dynamic arrays to store the primes and use this information in the mod test.

2. Write to the screen a list of the first 10000 primes in the format below. Report only the last five lines. Comment on the behaviour of the ratio $n * \ln(p(n))/p(n)$ as n gets large.

```
n   :  p(n)   : n*ln(p(n))/p(n)
1   :   2     : 0.34657359027997
2   :   3     : 0.73240819244541
3   :   5     : 0.96566274746046
```

3. Instead of writing to the screen, write to a file (on disk) a list containing just the prime numbers. Print eight numbers per line, such that all numbers have the same space, i.e., the file should start like:

```
2  3  5  7  11  ...
```

4. Time your code for $N = 10^3, 10^4, 10^5$ and 10^6. Make a log-log plot of run-time against N for both codes. What can we say from the log-log plot?
Hint: It should be possible to compute all these values.

5. How can the algorithm be optimized?

Exercise 3.

1. Write a program that reads in the prime number list, generated by your solution to exercise 2 (from file).
2. For each prime number, $p(n)$, compute $d(n)$, which is the number of digits in $s(n)$:

   $$s(n) = p(n) * p(n-1) * ... * p(1) + 1,$$

   e.g.

   $$s(3) = p(3) * p(2) * p(1) + 1 = 5 * 3 * 2 + 1 = 31,$$

   therefore

   $$d(3) = 2.$$

   Hint: You can use a recursive function to compute this.
3. Output to the screen n, $p(n)$ and $d(n)$ formatted in a suitable way.
4. No calculations should be done in the function `main`. Instead, break this problem into four functions, which
   (a) Compute the amount of data to allocate memory first.
   (b) Read the data from the file.
   (c) Compute and return an array d to the main program.
   (d) Output the information to the screen (using the arrays p and d).
   Note: Use pointers to pass the array between the different functions.
5. Do you run into problems as n gets large? If so, why? How can we avoid this problem?

Note: Exercise 4 is only required to obtain grades 7 and higher. A grade of 6 can be achieved without attempting this question.

Exercise 4: Address book.

Design a simple address book application using C++ structures to store the data. The book must hold 100 (or more) entries.

The following commands must be supported:

1. Add an entry to the address book.
2. Browse the address book.
3. Delete an entry from the address book.

4. Search for an entry in the book by any field of your choice.

When it is launched, the program will present a menu with the above choices, and will also display the number of entries in the book. The minimum fields for each entry are as follows:

1. Name (up to 64 characters)
2. Address (up to 128 characters)
3. Telephone Number (format checked by your program; (xxx)-xxx-xxxx)
4. Comment (up to 128 characters)

DESIGN CRITERIA: Each major component in the program must be contained in a separate function. Your main() function may contain a maximum of only 10 C++ statements. The program is not required to retain results between runs.

Note: Exercise 5 is only required to obtain grades 9 and 10. A grade of 8 can be achieved without attempting this question.

*Exercise 5: Heapsort.

A binary tree is a data structure in which every node has at most two children. A binary heap is a binary tree with two additional properties.

i The tree is complete: all levels are filled (contain data) except possibly the last elements of the deepest level.

ii The data of each node is smaller or equal to the data of each of its children.

The heapsort algorithm uses a binary heap to sort a list of number and consists of two steps. First all numbers are added in a special way to a binary heap (which you can store in an array of length N). After that has been done the elements can be taken out one by one, but now sorted in size.

For this question consider a sequence of N unsorted numbers; just take random numbers or the numbers $N, N-1, \ldots, 1$.

1. Construct a binary tree in C++.

2. Write an algorithm to add elements such that you always have a binary heap. The steps are

(a) Add a new element at the bottom of the heap.

(b) Compare the added element with its parents; if they are in the correct order, stop.

(c) If not, swap the element with its parents and return to step (b).

3. Write an algorithm which removes the smallest item and leaves your tree in a current heap state. The steps are

(a) Remove the root of the tree (this is the next smallest element).

(b) Replace the root of the heap with the last element in the lowest level.

(c) Compare the new root with its children; if they are in the correct order, stop.

(d) If not, swap the element with one of its children and return to step (c).

4. Test your implementation and use it to generate a sorted list. Compare the efficiency in the case that N is very large. Try to estimate how long it takes to sort a set of N points. Theoretically the time should be proportional to $N \ln N$ which in practice is clearly distinguishable from N^2 (take, e.g.,. $N = 10000$, $N = 100000$ and $N = 1000000$). Plot a log-log plot of run time against N and show that it is slower than order N, but quicker than order N^2.