Universität Zürich^UZH

Physik-Institut

Programmieren in C++
FS 2024

Lecture 4: Data handling in C/C++

1 Introduction to data handling

In the previous class, we went over how to process and analyse data within our code. In reality, most of the time the data we need to process cannot be hard-coded into a program. Most of the time we are given a set of data in a specific form and we have to import it into our variables in our code. You will not get an e-mail with a nice array or vector of integers, and you will not send one. You will more likely going to give or get a text file with many numbers, characters, strings and lines. Today, we will cover how we can import and output data using C/C++. Today's class is dedicated on how to import and prepare the data to be processed.

1.1 ifstream

Within C/C++ there is a built-in file-streaming function called "fstream". We need to load it at the beginning of the C++.

```
#include <fstream>
```

Now, suppose you are given a .txt data file as the following:

```
some_numbers.txt
1
2
3
4
5
6
7
8
9
10
11
12
13
10
```

Following a is a code to import the above data set. Inspect it to see what it does.

```
// data_import.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>

using namespace std;
int main(int argc,char *argv[]){ // Main begins
```

```cpp
        // Variable declaration
        ifstream input_data; // This is the variable that will be used to import data.
        input_data.open("some_numbers.txt");
        string line_holder; // This is a string that will hold onto the input line. Everything will be treated as a string.

        for (int i=0; i<10; i++){ // getline for loop
                getline(input_data,line_holder); // You can read one line at a time using getline.
                cout<<line_holder<<endl;
        } // getline for loop ends
        input_data.close(); // Always close your file as soon as you're done working with it.

        // Note that each time you get the line, you will have progressed to the next line, in order to read the file from the top again, you must close the ifstream and open the file again.


        cout<<"While method"<<endl;
        input_data.close(); // re-initializing the input file.
        input_data.open("some_numbers.txt");
        while (input_data >> line_holder){ // while input begins
                cout<<line_holder<<endl;
                // Here you are directly pushing each line from input_data into line_holder. No "getline" is needed.
                // While loop ends when the file ends as there is no more line to push into line_holder.
        } // while input ends.
        input_data.close();

    return 0;
} // Main ends
```

and the output:

```
some_numbers.txt
1
2
3
4
5
6
7
8
9
While method
some_numbers.txt
1
2
3
4
5
```

```
6
7
8
9
10
11
12
13
10
```

Note that while loop is the obvious choice for importing data especially when you do not know how long the data set is. Also, we know that the data other than the first line are integers. If we know that obviously we don't want to import them as strings, we want to import load them into an array or vectors.

```cpp
// data_import.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>

using namespace std;
int main(int argc,char *argv[]){ // Main begins
        // Variable declaration
        ifstream input_data; // This is the variable that will be used to
        ↪  import data.
        input_data.open("some_numbers.txt");
        string line_holder; // This is a string that will hold onto the
        ↪  input line. Everything will be treated as a string.

        // Removing the "some_numbers.txt in the header by getting the first
        ↪  line (typically headers are longer than 1 line)."
        for (int i=0; i<1; i++){ // getline for loop
                getline(input_data,line_holder);
        } // getline for loop ends

        // At this point we know that everything else in the file are
        ↪  integers, and we don't know how many integers there are. Let's
        ↪  create a vector of integers and load them.
        int dummy_integer;
        vector<int> input_integers;
        while (input_data >> dummy_integer){ // while input begins
        // Note that you still need the line_holder to allow the iteration.
                input_integers.push_back(dummy_integer);
        } // while input ends.
        input_data.close();

        // Now let's print and make sure we get what we imported
        int index=0;
        for (int element:  input_integers){
                printf("input_integers[%i]=%i \n",index,element);
                index++;
        }
    return 0;
} // Main ends
```

The output from the above code looks like the following:

```
input_integers[0]=1
input_integers[1]=2
input_integers[2]=3
input_integers[3]=4
input_integers[4]=5
input_integers[5]=6
input_integers[6]=7
input_integers[7]=8
input_integers[8]=9
input_integers[9]=10
input_integers[10]=11
input_integers[11]=12
input_integers[12]=13
input_integers[13]=10
```

As stated above, a typical header is more than just one line. It is important to inspect a given file and locate where the relevant data begins.
As most of you are also aware, a typical data set is not as simple and definitely not single dimensional. However the import method is just the same. Supposed you have a bit more complex file seen below.

```
1  some_data.txt
2  This is the header and it will start in the next couple of lines.
3  
4  
5  Or not.
6  x        y        z
7  1        2        87
8  2        4        99
9  3        7        33
10 4        3        11
11 5        2        634
12 6        9        213
13 7        11        41
14 8        31        532
15 9        312        12
16 10        56        76
17 11        13        31
18 12        89        321
19 13        137        12
20 10        1478        94
```

There are many different ways to deal with this header. The simplest way is to look at the line number and skip until line 7[1]. As for multi-dimensional data set, modify the while loop to accommodate more than a single element from each line.

```
// data_import.cxx
#include <iostream>
#include <stdio.h>
```

[1]in C index starts at 0, so skip until 6th element or index 6

```cpp
#include <vector>
#include <fstream>

using namespace std;
int main(int argc,char *argv[]){ // Main begins
        // Variable declaration
        ifstream input_data; // This is the variable that will be used to import data.
        input_data.open("some_data.txt");
        string line_holder; // This is a string that will hold onto the input line. Everything will be treated as a string.

        // Removing the "some_numbers.txt in the header by getting the first line (typically headers are longer than 1 line)."
        for (int i=0; i<6; i++){ // getline for loop
                getline(input_data,line_holder);
        } // getline for loop ends

        // At this point we know that everything else in the file are integers, and we don't know how many integers there are. Let's create a vector of integers and load them.
        int dummy_x, dummy_y, dummy_z;
        vector<int> x,y,z;
        // vector<int> input_integers;
        while (input_data >> dummy_x >> dummy_y >> dummy_z){ / /while input begins
        // Note that you still need the line_holder to allow the iteration.
                x.push_back(dummy_x);
                y.push_back(dummy_y);
                z.push_back(dummy_z);
        } // while input ends.
        input_data.close();

        // Now let's print and make sure we get what we imported
        int index=0;
        for (int element:  x){
                printf("x[%i]=%i, z[%i]=%i, z[%i]=%i \n",index,element,index,y[index],index,z[index]);
                index++;
        }
    return 0;
} // Main ends
```

As you can see, for data separated by tab or space are very easy to handle. Now let's move onto writing some data.

1.2 ofstream

Similar to ifstream, you must import fstream libraries in order to use ofstream. Ofstream is extremely similar to using a cout command. The following is an easy example. Let's reuse the random number generator from a previous example to output some data.

```cpp
// data_export.cxx
#include <iostream>
```

```cpp
#include <stdio.h>
#include <vector>
#include <fstream>
using namespace std;
int main(int argc,char *argv[]){ // Main begins
        // Define the threshold for the random number generation.
        const int rand_threshold=999999; // constant integers cannot be changed once declared.
        vector<int> numbers; // empty numbers vector
        int sign_generator=1; // a sign generator to assign + or -
        for (int i=0;i<1000000;i++){ // random vector generator begins
                sign_generator=-1; // by default the sign is negative
                if(rand()%2==0){ // roll the dice, if even = positive number. Statistically this is 50% of the numbers generated
                        sign_generator=1; // 50% of the numbers will be positive
                } // sign generator ends
                numbers.push_back(sign_generator* (rand() % rand_threshold)); // new random number generated and signed
        } // random numbers created
        // variables for linear search
        int max=-99999;
        int min=99999;
        vector<int> target_index;
        for (int value : numbers){ // Linear search begins
                if (value > max){ // max if statement begins
                        max=value;
                } // max found if statement ends
                if (value < min){ // max if statement begins
                        min=value;
                } // max found if statement ends
        } // Linear search ends
        cout<<"max is: "<<max<<endl;
        cout<<"min is: "<<min<<endl;

        // Now, let's create a header for it and write all of the random numers into a text.
        ofstream output;
        output.open("random_numbers.txt");

        output<<"random_numbers.txt"<<endl;
        output<<"max is: "<<max<<endl;
        output<<"min is: "<<min<<endl;

        for (int value : numbers){ // Linear search begins
                output<<value<<endl;
        } // Linear search ends
        output.close();
    return 0;
} // Main ends
```

The output file should look similar to this:

```
random_numbers.txt
max is: 999998
min is: -999995
-931732
-638629
-238759
...
```

You will see a great similarity of usage of ofstream as the cout. Let's now output some 3D data with minimal header.

```cpp
// data_export_3D.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>
using namespace std;
int main(int argc,char *argv[]){ // Main begins
        // Define the threshold for the random number generation.
        const int rand_threshold=999999; // constant integers cannot be
         ↪ changed once declared.
        ofstream output;
        output.open("random_numbers_3D.txt");
        output<<"random_numbers_3D.txt"<<endl;
        output<<"x \t y \t z"<<endl;
        int sign_generator=1; // a sign generator to assign + or -
        for (int i=0;i<100000;i++){ // random vector generator begins
                sign_generator=-1; // by default the sign is negative
                if(rand()%2==0){ // roll the dice, if even = positive
                 ↪ number. Statistically this is 50% of the numbers
                 ↪ generated
                        sign_generator=1; // 50% of the numbers will be
                         ↪ positive
                } // sign generator ends
                output<<sign_generator* (rand() %
                 ↪ rand_threshold)<<"\t"<<sign_generator* (rand() %
                 ↪ rand_threshold)<<"\t"<<sign_generator* (rand() %
                 ↪ rand_threshold)<<endl;
        } // random numbers created
        output.close();
    return 0;
} // Main ends
```

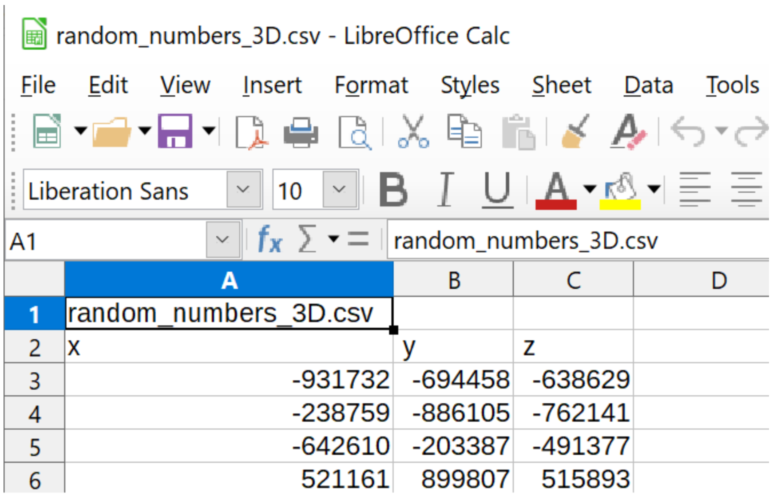
The resulting data should look something similar to:

```
random_numbers_3D.txt
x          y          z
-931732        -694458        -638629
-238759        -886105        -762141
-642610        -203387        -491377
521161        899807        515893
384966        89476        457039
...
```

To be able to easily access the data from MS Excel or LibreOffice Calc, a common data file type used is ”comma-separated values” .csv. You can create this easily by modifying above code to the following.

```cpp
// data_export_3D_csv.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>
using namespace std;
int main(int argc,char *argv[]){ // Main begins
        // Define the threshold for the random number generation.
        const int rand_threshold=999999; // constant integers cannot be
        ↪ changed once declared.
        ofstream output;
        output.open("random_numbers_3D.csv");
        output<<"random_numbers_3D.csv"<<endl;
        output<<"x , y , z"<<endl;
        int sign_generator=1; // a sign generator to assign + or -
        for (int i=0;i<100000;i++){ // random vector generator begins
                sign_generator=-1; // by default the sign is negative
                if(rand()%2==0){ // roll the dice, if even = positive
                ↪ number. Statistically this is 50% of the numbers
                ↪ generated
                        sign_generator=1; // 50% of the numbers will be
                        ↪ positive
                } // sign generator ends
                output<<sign_generator* (rand() %
                ↪ rand_threshold)<<","<<sign_generator* (rand() %
                ↪ rand_threshold)<<","<<sign_generator* (rand() %
                ↪ rand_threshold)<<endl;
        } // random numbers created
        output.close();
    return 0;
} // Main ends
```

The output should look similar to the following: Note that reading .csv files is not very trivial


random_numbers_3D.csv - LibreOffice Calc
File Edit View Insert Format Styles Sheet Data Tools
Liberation Sans
10
A1

random_numbers_3D.csv

	A	B	C	D
1	random_numbers_3D.csv			
2	x	y	z	
3	-931732	-694458	-638629	
4	-238759	-886105	-762141	
5	-642610	-203387	-491377	
6	521161	899807	515893	

Figure 1: Output .csv file from the above code.

in C++. It involves reading data line-by-line then cut by the line by ”comma” separators[2] as strings then converted into integers.

[2]called delimiter

= The practical programming part of this course will now begin for 60 minutes. =

2 Creating and reading files

1. Create each of the following using C/C++ code with at least 10 lines of headers. The header must include:
   - Maximum and minimum of the elements.
   - Average of the elements.
   - Median of the elements.
   - Skew of the elements.
   - Sample standard deviation of the elements.
   - Total number of elements.
   - Dimension of the data.
   - Title of the data.
   - Data file name and type.
   - Date and time of the random number generation.

   - A 1 dimensional data set with 1000 random number elements.
   - A 2 dimensional data set with tabular separation and 10000 random number elements.
   - A 3 dimensional data set with tabular separation and 10000 random number elements.
   - Same as above but into comma separation and into a .csv file.
2. In your functions package, write 3 functions to read each of the above[3].
3. Create a function that will allow you to save an array into a file.
4. Create a function that will allow you to save a vector into a file.

[3]Except for the .csv file since it has not been covered yet

= The theoretical lecture part of this course will now continue for 15 minutes. =

3 Typecasting

As mentioned before, working with a .csv file as an input file is not very trivial in C++. Suppose you are trying to read the file "random_numbers_3D.csv" seen in **Figure 1** . Since you are aware that each line has to be loaded as strings then cut by the delimiters, you need to import another package called stringstream or sstream. You can do this by writing:

```
#include <sstream>
```

Below you will find a good example of using stringstream from reading a .csv file.

```
// read_csv.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>
#include <sstream>
using namespace std;
int main(int argc,char *argv[]){ // Main begins
        ifstream input;
        input.open("random_numbers_3D.csv");
        string temp_string;
        for (int i=0 ; i < 2; i++){
                getline(input,temp_string);
        } // Headers have been read and skipped

        vector<int> x,y,z;
        int counter=0;
        char delimiter=','; // comma separator.
        while( input >> temp_string  ){ // input while loop starts
                stringstream temp_sstream(temp_string);//stringstream
                ↪  created then destroyed at the end of each iteration in
                ↪  stack.
                while( getline(temp_sstream, temp_string, delimiter) ){ //
                ↪  Seperating each line by the delimiter.
                        counter=counter%3;
                        switch (counter){ // Switch case is used to
                        ↪  determine x y or z
                                case 0: // This is x
                                x.push_back(stoi(temp_string.c_str()));
                                break;
                                case 1: // This is y
                                y.push_back(stoi(temp_string.c_str()));
                                break;
                                case 2: // This is z
                                z.push_back(stoi(temp_string.c_str()));
                                break;
                        } // switch case ends
                        counter++;
                } // stringstream while loop ends
        } // input while loop ends

        for (int i=0; i<10; i++){
                printf(" %i , %i, %i  \n", x[i],y[i],z[i]);
        }

        input.close();
    return 0;
} // Main ends
```

In the above case, you will notice that the stringstream is being called temporally at the beginning of each while iteration. This is safe as we know that we are creating and destroying a temp_sstream in the beginning of the while iteration and it gets destroyed at the end of each iteration. This is because we are using stack part of the memory and we do not anticipate the stringstream holding large data. Please note that you should never call a new heap variable inside[4].
You will also notice the usage of ”stoi” string to integer function. You may also have noticed the usage of ”c_str()”. These are two different functions. The stoi conversion function is part of the standard C++ library which can convert character-strings to integers. However as described above, it does not directly accept a c++ string type. It instead accepts a character-string as this package has been built on top of character-string. The difference between c-string and a c++ ”string” is that the programming language C, there is no such thing as ”string”. Instead you can create an array of characters and treat them as a string of characters. You will find more information on these functions in the following link https://en.cppreference.com/w/cpp/string/basic_string/stol.
The important part for you in this course is that you learn how to use them. The output of the above code is as follows:

```
-931732 , -694458, -638629
-238759 , -886105, -762141
-642610 , -203387, -491377
521161 , 899807, 515893
384966 , 89476, 457039
-595889 , -702861, -958155
22391 , 723140, 665356
-703603 , -515030, -981603
723693 , 134438, 899292
-20545 , -175639, -479698
```

As you may have noticed, .csv is not the only filetype that will require you to import data non-trivially. There are many data types, data corruptions and encryptions which require the manipulation of each line and its subset.
In the following practical session, we will practice handling data of different types.

[4] heap and stack are memory management terms and will be covered in the next class

= The practical programming part of this course will now begin for 60 minutes. =

4 Data handling practice

1. Recall the very last element of exercise 1. Create a function that will read the .csv file you've created, and compare the output with an LibreOffice Calc[5].
2. Create a general function that intakes a delimiter character input and a stringstream and prints each element.
3. Create a general function that intakes a delimiter character input and a string and prints each element.
4. Write a program that can perform Caesarian encryption and decryption (for all 26 keys). For more information on Caesarian encryption please visit http://practicalcryptography.com/ciphers/caesar-cipher/#:~:text=The%20Caesar%20cipher%20is%20one,become%20C%2C%20and%20so%20on..

5 Conclusion

You have now learned the basics of data handling. Not only can you now analyse the data, now you have the capability of importing and exporting many different types of data regardless of how the data is organized. You must understand that even up until this point we cannot handle very large data efficiently.
In most inefficient programs, the data gets written and read from the storage device multiple times. This is typically very slow especially if the machines are equipped with tape or platter hard drives.
For efficient and fast programs, the data is read once from the hard drive, and kept in RAM until terminated. As long as your machine has enough RAM, you should never use the hard drive other than to save the results of the analyses. Up until now, we have been only using a small section of the RAM called "stack". In the next course we will start using the larger section of the RAM called "heap".

[5] or MS Excel