Universität Zürich[UZH]
Physik-Institut

Lecture 6: Handling large data in C/C++

1 Handling large data in C/C++

Now that we know how to handle and process data in heap and stack, we will now practice handling large data. For the most of today's lecture, we will go through the exercises and practice good programming. Since we will be using what we've learned, there will be very little theory.

1.1 Sampling

Before you start handling very large data, you want to make sure that your code actually works in small samples. For example, if you want to create and handle 10 large number of integers see the following example.

```cpp
// declare_large.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>
#include <sstream>
using namespace std;
void check(char c=0){
        cout<<"Crash point "<<c<<endl;
}
int main(int argc,char *argv[]){ // Main begins
        check();
        int a[1000];
        check('a');
        int *e=(int*)malloc(1000000000*sizeof(int));
        check('e');
        int *f=(int*)malloc(10000000000*sizeof(int));
        check('f');
        free(e);
        free(f);
    return 0;
} // Main ends
```

be sure to compile and execute and look for the segmentation faults.

```
Segmentation fault (core dumped)
```

This is a blessing. This means that your code compiles but fails to run and the computer knows that it failed to work. Always compile and run and check before and after you make changes to your code.
Suppose that now you want to work with *f. You want to make sure you only sample a small section of it. For example:

```cpp
// declare_large.cxx
#include <iostream>
#include <stdio.h>
#include <vector>
#include <fstream>
#include <sstream>
using namespace std;
int main(int argc,char *argv[]){ // Main begins
        int max=1000000000;
        int *f=(int*)malloc(max*sizeof(int));

        // random number assignment.
        for (int i=0; i<max;i++){
                f[i]=rand();
        }

        // Checking for the assigned numbers
        for (int i=0; i<max;i++){
                if(&f[i]!=NULL){
                        if (i<20) cout << f[i] << endl;
                        // cout << i << ":\t" << f[i] << endl;
                }
        }

        free(f);
        return 0;
} // Main ends
```

You can actually execute this file and look at the memory usage similar to **Figure 1** . It is very important to make sure that when your code has finished reading, it releases all of the memory back to the operating system. You also need to make sure that there is no run-away-in memory so that your computer runs out of memory before the program has completed all of its tasks.
The output of above code should be:

```
1804289383
846930886
1681692777
1714636915
1957747793
424238335
719885386
1649760492
596516649
1189641421
...(There will be a pause until your computer checks every single available
↪  element in your *f for non-null value.)
```

As soon as that you do not need a certain segment of memory, free the memory and if necessary delete the pointer.

Also when you are sampling your code, be sure to be ready to kill the process. If there is a memory run-away scenario as you will crash your computer.

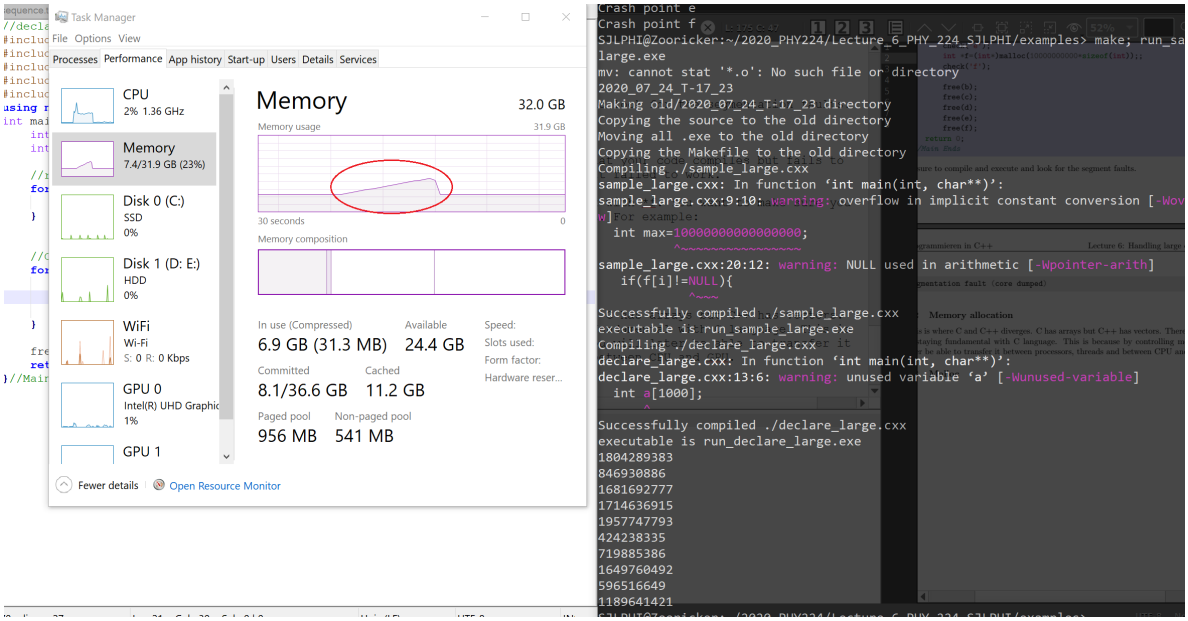

Figure 1: The memory usage is highlighted and peaks at the end of the execution of the above code.

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

2 Practice session 1

1. Look up and calculate the maximum number of the following in 1.5 GB:
   - integer
   - short integer
   - long long integer
   - bool
   - string
   - char
   - float
   - double
   - double double [1]
2. Create a program that declares, assigns, frees and deletes the maximum number of each type above (make sure using less than 1.5 GB).
3. Create a random boolean generator that will produce an outcome of true or false.
4. Create a function in the most efficient way possible to create and store the maximum number of random true/false values.
5. Same as above but using unsigned integer.
6. Using the above two, create a function that will allow you to use the boolean as the sign of an integer. In short, create a way to store and use 5-byte signed "unsigned" integer.
7. Create a 1 GB 2 dimensional integer array and input all random unsigned integers.
8. Create a function to calculate the mean, median, sample standard deviation, skew, maximum and minimum of the above in each dimension.

[1] This data type might not work for Mac

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

3 Freeing memory

In the following practice session we will focus on declaring then destroying large variables. Look at the following example

```
// declare_and_destroy.cxx
#include<iostream>
#include<stdio.h>
#include<vector>
#include<fstream>
#include<sstream>
using namespace std;
int main(int argc,char *argv[]){ // Main begins
        int max=1000000000;
        int *f=(int*)malloc(max*sizeof(int));
        // random number assignment.
        for (int i=0; i<max;i++){
                f[i]=rand();
        }
        // Checking for the assigned numbers
        for (int i=0; i<max;i++){
                if(&f[i]!=NULL){
                        // cout << f[i] << endl;
                }
        }
        free(f);
        int *g=(int*)malloc(max*sizeof(int));
        // random number assignment.
        for (int i=0; i<max;i++){
                g[i]=rand();
        }
        // Checking for the assigned numbers
        for (int i=0; i<max;i++){
                if(&g[i]!=NULL){
                        // couti << g[i ]<< endl;
                }
        }
        free(g);

    return 0;
} // Main ends
```

You will notice that the memory allocation happens twice in the above code. You should also notice that the memory freed up in between. You can physically see the usage similar to **Figure 2** .

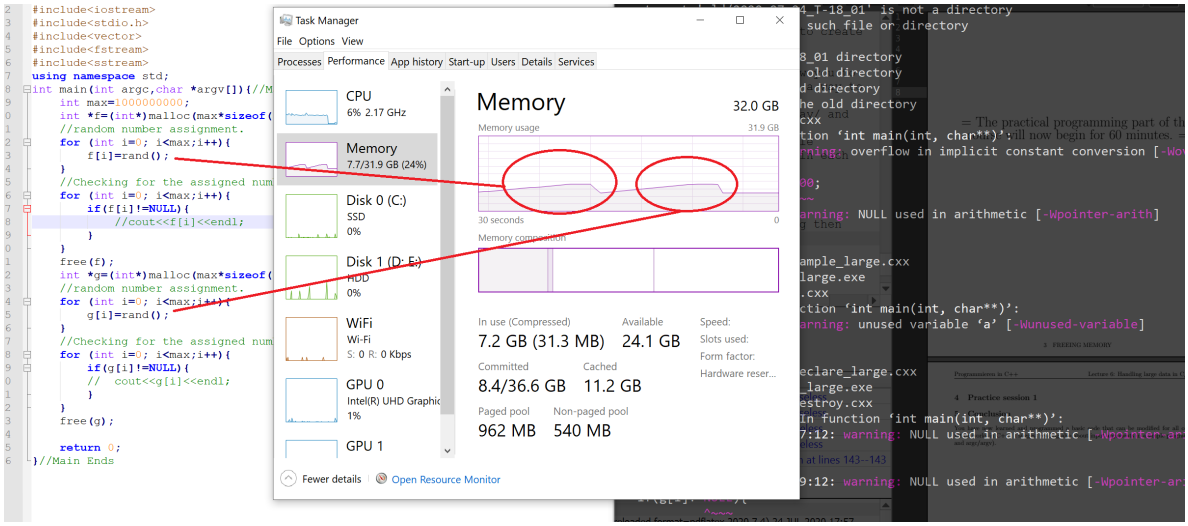

Figure 2: You can see when the memory is being used and freed up within the program.

It is now time to practice this.

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

4 Practice session 2

1. Create two 0.5 GB 2 dimensional integer arrays (positive and negative).
2. Create a general function to calculate the mean, median, sample standard deviation, skew, maximum and minimum of the above in each dimension.
3. Create a function to perform elemental addition, subtraction and multiplication of two 2d arrays.
4. Modify the above function to store the results and free up the memory of the two arrays used in calculation.
5. Create a 1 GB random +/- integer vector.
6. Create a function that intakes the above vector, multiplies each element by a factor of 3 and outputs a new vector while destroying and freeing the old one.

5 Conclusion

You now have all of the tools to perform data analysis in general sense. However, in future you will most likely not have the opportunity to write a short code. You will most likely going to have to work in groups. When we work in groups, it is best to assign everyone on one small item rather than everyone working out of a single large code.
This is one of the greatest advantage of object-oriented programming, everyone can work on many different things at the same time by cutting up parts of the code into smaller.
In the next class we will go over objects and classes.