Universität Zürich[UZH]
Physik-Institut

Lecture 3: Data processing in C/C++

1 Introduction to data processing

Up until now, we have been learning how to use some the very basic tools available to create a program. Today we will focus on how to build programs process and analyse data. We will however stay with simple small sets of data which will fit in "stack" and also only operate using RAM.
Now that we know how to use if statements and for loops, we need to go through how to use them to process data. Today's session will cover linear search and the escape conditions from within a loop and switch cases.

2 Linear search

The term "linear search" is dedicated to searching linearly throughout a set of data. A linear search looks at each and every element in a given set of data. A simplest case is when we are searching for a match of a target value linearly.

2.1 Match

The following is one of the simplest examples using a for loop:

```
int some_integers[13] = {1,2,3,4,5,6,7,8,9,10,11,12,13};
int target = 10;
int target_index;
for (int i = 0; i < 13; i++){
        if (some_integers[i]==target){
                cout<<"The target is found and it has the index "<<i<<endl;
                target_index=i;
        }
}
```

The above is obviously terribly optimized and has a lot of room for improvement. Furthermore, it becomes useless as soon as there are more than one element in the array that has the same target value.

```
int some_integers[14] = {1,2,3,4,5,6,7,8,9,10,11,12,13,10};
int target = 10;
int target_index;
for (int i = 0; i < 14; i++){
        if (some_integers[i]==target){
                cout<<"The target is found and it has the index "<<i<<endl;
                target_index=i;
        }
}
```

The above will let you know where the target values are but it will only save one index. We can fix this rather easily by making the target index to be an array or vector. In this case, a

vector is a better choice since we don't know how many values in the array have the target value.

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

using namespace std;
int main(int argc,char *argv[]){//Main begins
        //Variable declaration
        int some_integers[14] = {1,2,3,4,5,6,7,8,9,10,11,12,13,10};
        int target = 10;
        vector<int> target_index;

        for (int i = 0; i < 14; i++){//Linear search begins
                if (some_integers[i]==target){//Target found if statement
                ↪  begins
                        cout<<"The target is found and it has the index
                        ↪  "<<i<<endl;
                        target_index.push_back(i);
                }//Target found if statement ends
        }//Linear search ends

        //Check to make sure that all of the indices have been saved
        ↪  properly.
        for (int i : target_index){//Checking for loop starts
                cout<<i<<endl;
        }//Checking for loop ends

    return 0;
}//Main Ends
```

The is not limited to integers. The above can be used for searching for matches in strings, characters, and any kind of property that can be programmed in. For example if you have an array of strings with what kind of pet you have at home, you can search for how many people and who has a "dog"

```cpp
//linear2.cxx
#include<iostream>
#include<stdio.h>
#include<vector>

using namespace std;
int main(int argc,char *argv[]){//Main begins
        //Variable declaration
        string owner[14] =
        ↪  {"Bob","Donald","Alice","Elodie","Chris","Johannes",
        "Roland","Micael","Ben","Ashley","Paul","Steven","Peter","Alvaro"};
        string pets[14] =
        ↪  {"dog","cat","lion","dog","bird","turtle","hamster",
        "dog","guinea pig","pig","fish","dog","rat","cat"};
        string target = "dog";
        vector<int> target_index;
```

```

        for (int i = 0; i < 14; i++){//Linear search begins
                if (pets[i]==target){//Target found if statement begins
                        cout<<"The "<<target<<" is found and it has the
                        ↪  index "<<i<<endl;
                        target_index.push_back(i);
                }//Target found if statement ends
        }//Linear search ends

        //See who owns a dog
        for (int i : target_index){//Checking for loop starts
                cout<<owner[i]<<" has a "<<target<<endl;
        }//Checking for loop ends

    return 0;
}//Main Ends
```

2.2 Minimum and maximum

There are other types of linear searches. Following is an example of linear search for the maximum and minimum value in a given array. Inspect the following code:

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

using namespace std;
int main(int argc,char *argv[]){//Main begins
        //Variable declaration
        vector<int> numbers={1,2,3,54,1,532,14,3,14,31,4,321,
        5,35,21,5,3215,324,324,321,4,3214,321,43,14,32,14,32,
        14,32,41,32,432,1,432,14,321,43,24,321,4,3214,32,431,35,
        324321,161,6,17,34121,7,78,53,45,24,-3143,35,432,65,437,321};
        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;

    return 0;
}//Main Ends
```

The output should be the following.

```
max is: 324321
min is: -3143
```

Note that the min and max are initialized to 999999 and -999999 respectively before the linear. This is because when you are performing a linear search for minimum and maximum you the computer can only achieve that by comparing the current value to the reference value.
Since in the beginning you do not know how big the maximum value is, it is best to give it a very low reference number such as -999999. Similarly, since you do not know in the beginning how small the minimum value is, it is best to give it a large value such as 999999.
Here's another example:

```cpp
//rand_minmax.cxx
#include<iostream>
#include<stdio.h>
#include<vector>
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;
    return 0;
}//Main Ends
```

Now, let's process some data using linear search.

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

3 Improving your functions package

– Modify your functions package from our previous session so that it has the following:

1. Build a random number generator function that can intake upper and lower limits and assign a signed (±) random number. See http://www.cplusplus.com/reference/cstdlib/rand/ and the last example for reference.
2. A linear search function that can return an exact match in an array (one for string and one for integer at least).
3. A linear search function that can return a minimum value in an array (float).
4. A linear search function that can return a maximum value in an array (float).
5. A linear search that can return the following from an array of numbers:
   (a) Minimum and how many times they appear.
   (b) Maximum and how many times they appear.
   (c) Median and how many times they appear ($\hat{\mu}$).
   (d) Sum of all numbers.
   (e) Average ($\bar{\mu}$).
   (f) Skewness ($\bar{\mu} - \hat{\mu}$)
   (g) Sum of all square of numbers.
   (h) Population variance ($\sigma^2 = \frac{1}{N}\sum_i^N x_i^2 - (\frac{1}{N}\sum_i^N x)^2$) and standard deviation (σ).
   (i) Sample variance ($s^2 = \frac{1}{N-n_{dof}}\sum_i^N (x_i - \bar{x}_n)^2$) and standard deviation (s) (with 1 degree of freedom (n_{dof}=1)).
   (j) How many numbers in $\bar{\mu} \pm \sigma$, $\bar{\mu} \pm 2\sigma$ and $\bar{\mu} \pm 3\sigma$

– Test all of the above using a vector with 10000 random floats in the range (-999999,999999).

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

4 Data sorting

As you may have found in the last exercise, from a single for loop many operation can be done. Within some libraries you can always use built-in functions such as finding the average, median, max/min and some other properties. You need to keep in mind that each time you use such function, the function has to loop over the entire array or vector. For 1000000 integers, our computers can do such operation in very short period. You can use a built in bash timer in the following manner to check:

```
time run_rand_minmax.exe
max is: 999998
min is: -999995

real    0m0.144s
user    0m0.031s
sys     0m0.047s
```

the results from increasing from 1000000 to 100000000 numbers result in the following time chart:

```
max is: 999998
min is: -999998

real    0m5.183s
user    0m4.125s
sys     0m0.984s
```

There is a way to calculate the approximate computational power required for each operation but we will not cover that in this course. However what you need to keep in mind is that each time you call a built-in function that requires looping over all of your elements in an array or vector, it wastes the computational time accessing each element. If you program your code in a way where multiple operations can be done from a single loop, it is more efficient.
For example, you should try to write your own function to get max, min, average, standard deviation, median and etc. in a single loop rather than running one loop to look for a max, one loop for min and etc.
For the duration of this course, we are not dealing with such large data, but depending on your field and your application, you can be dealing with data set much larger than 100 million integers.
Something else we need to pay attention to is that most of the time we need to categorize and sort our data. In some cases this is called binning, categorization or even clustering. Let's look at the following example:

```
1  //switch_case0.cxx
2  #include<iostream>
3  #include<stdio.h>
4  #include<vector>
5  using namespace std;
6  void checker(int i= (int)(NULL)){
7          if(i!=(int)(NULL)){
8                  cout<<"I got "<<i<<endl;
9          }
10         else{
11                 cout<<"I got NULL"<<endl;
12         }
13 }
```

```
int main(int argc,char *argv[]){//Main begins
        int input=2;
        if (input==1){checker(1);}
        if (input==2){checker(2);}
        if (input==3){checker(3);}
        if (input==4){checker(4);}
    return 0;
}//Main Ends
```

the output is the following:

```
I got 2
```

The above code will look for the condition when input is 1, 2, 3 or 4. As you can see for every condition, there is an entirely new if statement. This is tedious and annoying. Also, what if we want to perform a hardware operation in sequence?
This is where ”switch” case becomes useful

4.1 Switch case

The following has the same result as above but using switch case condition:

```
//switch_case0.cxx
#include<iostream>
#include<stdio.h>
#include<vector>
using namespace std;
void checker(int i= (int)(NULL)){
        if(i!=(int)(NULL)){
                cout<<"I got "<<i<<endl;
        }
        else{
                cout<<"I got NULL"<<endl;
        }
}
int main(int argc,char *argv[]){//Main begins
        int input=2;
        switch (input){
                case 1: checker(1);
                break;
                case 2: checker(2);
                break;
                case 3: checker(3);
                break;
                case 4: checker(4);
                break;
        }
    return 0;
}//Main Ends
```

output:

```
I got 2
```

Notice that the "breaks" have to be placed in between each case. Also the cases do not have to be in order. Let's look at a small variation of it:

```
//switch_case0.cxx
#include<iostream>
#include<stdio.h>
#include<vector>
using namespace std;
void checker(int i= (int)(NULL)){
        if(i!=(int)(NULL)){
                cout<<"I got "<<i<<endl;
        }
        else{
                cout<<"I got NULL"<<endl;
        }
}
int main(int argc,char *argv[]){//Main begins
        int input=2;
        switch (input){
                case 2: checker(2);
                case 1: checker(1);
                case 4: checker(4);
                break;
                case 3: checker(3);
                break;
        }
    return 0;
}//Main Ends
```

and the result is:

```
I got 2
I got 1
I got 4
```

You may wonder why this is important. This is particularly useful when you are sorting items as seen in the following example:

```
//switch_case2.cxx
#include<iostream>
#include<stdio.h>
#include<vector>

using namespace std;

constexpr unsigned int sitranslate(const char* input_string, int char_index
 ↪  = 0)
{
    return !input_string[char_index] ? 5381 : (sitranslate(input_string,
     ↪  char_index+1) * 33) ^ input_string[char_index];
}
int main(int argc,char *argv[]){//Main begins
        //string animal="dog";
        vector<string> animals={"dog","mouse","rodent","pet","domesticated
         ↪  animal","dog"};
```

```
15
16          for (string animal : animals){
17                  int input=sitranslate(animal.c_str());
18                  switch (input){
19                          case sitranslate("dog"): cout<<"This is a dog, ";
20                          case sitranslate("pet"): cout<<"This is a pet, ";
21                          case sitranslate("domesticated animal"): cout<<"This
                             ↪  is a domesticated animal";
22                          cout<<endl;
23                          break;
24                          case sitranslate("mouse"): cout<<"This is a mouse,
                             ↪  ";
25                          case sitranslate("rodent"): cout<<"This is a
                             ↪  domesticated animal";
26                          cout<<endl;
27                          break;
28                  }
29          }
30      return 0;
31  }//Main Ends
```

and the output is:

```
This is a dog, This is a pet, This is a domesticated animal
This is a mouse, This is a domesticated animal
This is a domesticated animal
This is a pet, This is a domesticated animal
This is a domesticated animal
This is a dog, This is a pet, This is a domesticated animal
```

Similarly such case can be applied to computer I/O. This is particularly useful for all programs and hardware that requires human-interaction because it allows us to control our interactions with the computer.
Also note that there is a string-to-integer translator function written. This is because C++ does not yet have a proper interpreter by default to accept strings in a case scenario. For more information there is a stack exchange discussion in the following: https://stackoverflow.com/questions/2111667/compile-time-string-hashing but this is far beyond the scope of this course and you do not need to know much about it.
The following is an example of using switch case to map keys for a stereotypical computer game ”arrow” and ”w”,”a”,”s”,b” keys and exit keys.

```
1   //switch_case3.cxx
2   #include<iostream>
3   #include<stdio.h>
4   #include<vector>
5   using namespace std;
6
7   #include<curses.h> //You must compile with -lncurses flag after your object
     ↪  is called
8   //Something like this:
9   //g++ -o switch_case3.exe -Wall switch_case3.cxx -lncurses
10  //Defining the key presses to integers so that it can be called from case
11  #define KEY_UP 65
12  #define KEY_DOWN 66
```

```
#define KEY_LEFT 68
#define KEY_RIGHT 67

int main(int argc,char *argv[]){//Main begins
  initscr(); //initiating screen for ncurses library
  noecho(); //eliminating echoing of the key presses in the screen
  int hit_key = 0; //initializing key press
  bool loop=true; //escape condition for the while loop
while(loop){
        switch(hit_key=getch()){
                case 'x' :
                case 'q' :
                cout<<"Pressed e(x)it or (q)uit key";
                cout<<"aborting"<<endl;
                //endwin();
                //return(0);
                loop=false;
                break;
                case KEY_UP:
                case 'w':
                cout<<"pressing up"<<endl;
                break;
                case KEY_DOWN:
                case 's':
                cout<<"pressing down"<<endl;
                break;
                case KEY_LEFT:
                case 'a':
                cout<<"pressing left"<<endl;
                break;
                case KEY_RIGHT:
                case 'd':
                cout<<"pressing right"<<endl;
                break;
        }
}
        endwin(); //returning the screen to normal before exiting.
    return 0;
}//Main Ends
```

Be mindful that the above uses a new library called "libncurses". You will not be able to compile unless if you set the flag "-lncurses" in the following manner.

```
g++ -o switch_case3.exe -Wall switch_case3.cxx -lncurses
```

or in your Makefile

```
inputcode=switch_case3.cxx
CPP=g++
SFLAG=-Wall
PFLAG=-lncurses
@$(CPP) -o $(addprefix run_,$@.exe) $(SFLAG) $(intputcode) $(PFLAG)
```

5 Escape conditions

As you have seen, the switch cases and while loops typically work together to work as an interactive program. If you are wondering practically all of your devices including your computer, phone, remote control, mouse, or even car are in an infinite loop listening for "case" switches.
However you do not want to also switch your computers, phones, remote control or cars off. If we take the last example, recall the lines 20 to 30

```
1  int hit_key = 0; //initializing key press
2    bool loop=true; //escape condition for the while loop
3  while(loop){
4          switch(hit_key=getch()){
5                  case 'x' :
6                  case 'q' :
7                  cout<<"Pressed e(x)it or (q)uit key";
8                  cout<<"aborting"<<endl;
9                  //endwin();
10                 //return(0);
11                 loop=false;
12                 break;
```

and also the lines 49 and 50

```
1          endwin(); //returning the screen to normal before exiting.
2      return 0;
```

You may notice that this case will switch the boolean loop to false, so that the while loop will terminate, then "breaks" out of current iteration. The following is a summary of each escape conditions.

```
1  break; //will take you out of the current iteration.
2  return; //will take you out of the current function. In the above case the
    ↪  function is main().
3  exit(0); //will end the program completely.
```

Note that in switch_case3.exe I always call "endwin();" before escaping out of the main. With the "endwin();" your terminal will be broken. In case if you've tried, you can fix it by the following command line (although you will not be able to see what you type):

```
reset
```

If you are not careful with the escape conditions, you will very easily break your terminal session and sometimes the operating system resulting in system freeze and crash. Therefore you must make sure all of your commands have an escape condition (otherwise it will be stuck in the infinite loop) and/or break. I have not yet observed a computer hardware breaking as a result, but a bad program will cause malfunctions in cars, phones and any hardware that can cause serious damage. So let's practice our escape conditions.

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

6 Sorting data

– Modify your functions package from our previous session so that it has the following:

1. Create an integer function that takes in an array of integers as input and using the switch case sorts them by multiples (modulus %) of 2, 3, 5, 7 and 11 and prints them on the screen in the following manner:

2	3	5	7	11
4	6	5	14	33
2	3	10	7	11
10	24	15	21	22

   within a for loop
2. Do the same as above using vectors within an infinite while loop.
3. Improve the second function to include negative numbers.
4. Improve the last function above to escape from the while loop (break;) when an instance of 2020 has been input.
5. Improve the last function above to escape from the function (return;) when an instance of 6666 has been input.
6. Improve the last function above to end program (exit(0);) when an instance of 12321 has been called.
7. Improve your main, and your functions to package to output that it is escaping from a loop, function or program.
8. Create a vector of 10000 random numbers and feed it through the function.

7 Conclusion

You now have all of the tools to process data and analyse a given set of information. All of what we've covered are very important[1] for this course, and any type of data analysis.
What we are missing so far is data handling. We have not yet talked at all about how to intake, and output data from and into files.
The next class will cover data handling where we import and export data from the computer's data drive and use smaller portion of memory called "stack" to perform analyses. Often, mishandling of memory leads to "memory leak". The advantage of using "stack" is that when the program terminates, the "stack" gets returned to the operating system.
However the disadvantage of "stack" are numerous. For more information on memory please visit: https://www.geeksforgeeks.org/stack-vs-heap-memory-allocation/#:~:text=Insertion%20Sort-,Stack%20vs%20Heap%20Memory%20Allocation,allocated%20on%20stack%20or%20heap.&text=Stack%20Allocation%20%3A%20The%20allocation%20happens,happens%20in%20function%20call%20stack. However the course on memory management will come after data handling session.
Please keep practising, and complete what you could not here today at home for tomorrow.

[1] with an exception to lines 8 to 11 of switch_case2.cxx