Universität Zürich[UZH]
Physik-Institut

Lecture 2: Introduction to basics of programming

1 Introduction to programming data types

In our previous class, we went over how to compile and write a basic C++ code in a linux environment. So far, what you've created is a a working platform that can be used for the rest of this block course. What we didn't go through in our previous class are data types. If you recall, your very last exercise and homework from last lecture was to use

```
int argc, char* argv[]
```

in your main code to input arguments at the execution of your program. Obviously this is very useful and essential to many programs. You should also note that the standard

```
int argc
```

stands for argument count integer and

```
char* argv[]
```

stands for a character pointer to the array of arguments. We will not talk about "pointers" today, but later on. The important part for Today is how to access and use these.

```
//minmainargs.cxx
#include <iostream>
#include <stdio.h>
using namespace std;
int main(int argc,char *argv[]){
        cout<<"Number of input arguments: "<<argc<<endl;
        for(int i=0; i<argc; i++){
                cout<<"argument count, argument: "<<i<<" , "<<argv[i]<<endl;
        }
    return 0;
}//Main Ends
```

The above uses what is called a "for" loop, which we will talk about later, what you need to keep in mind is that ALL computer indices start with zero and the "argument" start with the program itself. For example if you compile and execute:

```
run_minimainargs.exe blah
```

the return will be

```
Number of input arguments: 2
argument count, argument: 0 , run_minimainargs.exe
argument count, argument: 1 , blah
```

This is not very "human" friendly because our "argument" is what we input after the executable. We should therefore modify the code to:

```cpp
//minmainargs_mod.cxx
#include<iostream>
#include<stdio.h>
using namespace std;
int main(int argc,char *argv[]){
        cout<<"Number of input arguments: "<<argc-1<<endl;
        for(int i=1; i<argc; i++){
                cout<<"argument count, argument: "<<i<<" , "<<argv[i]<<endl;
        }
    return 0;
}//Main Ends
```

and the result:

```
run_minimainargs_mod.exe c++ is fun
Number of input arguments: 3
argument count, argument: 1 , c++
argument count, argument: 2 , is
argument count, argument: 3 , fun
```

Today, we will be talking about types, arrays, vectors, while loop, for loop, if statements and the logical operators. For the purpose of Today's lecture we will also only use "heap" memory.

1.1 Primitive data types

Types are classification of variables. The fundamental types are already built in within the C language and more in C++ libraries. We will start with types available in C.

```cpp
int
```

int is simply integers($1, 2, 3, 4, 5... \in \mathbb{Z}$). There are available prefixes to the integers such signed, unsigned, long, short which determine more specifically what integers you are using.

```cpp
float
```

float is simply floating point decimal number ($1.215, \pi, 54151.3141... \in \mathbb{R}$).

```cpp
char
```

char is simply a character type which stores an integer to represent a character using a numerical code such as American Standard Code for Information Interchange (ASCII). For example the capital letter A is assigned the integer 65 in ASCII.

```cpp
void
```

void means "nothing" in English and that just is it in programming as well. Voids return nothing. They are most useful for creating custom functions (called methods when inside an object).

The standard C++ package includes more primitive data types which makes your programming experience arguably easier. The standard C++ package in addition to existing C primitive data types will give you:

```
bool
```

bool is Boolean. It is truly a binary data type that holds the value either

```
true
```

or

```
false
```

```
double
```

double is floating point decimal number but double in data size.

```
wchar_t
```

wchar_t is wide character, which is just character but double or quadruple in data size.

```
string
```

string is a string of characters. In primitive C, a string is typically defined as an array of characters.

There are also prefixes such as "constant", "unsigned", and "signed" variables but they will not be covered within this course.
One of the greatest things about object oriented programming is that you can create your own data type, however today we will only work with the above standard data types. You can declare them in the following manner:

```
int one = 1;
char two = 't';
float three = 1234.56789;
void four();
wchar_t five = 21321;
string six = "six";
double seven = 1234.56789;
bool eight = true;
```

Also keep in mind that you do not have to specify the variables that you declare.

1.2 Arrays

If you have many of the same objects of the same type such as characters, you can create a larger set of data with them. These are called derived data types. The standard C language gives us an array. Below are three different ways of declaring arrays:

```
int one_to_five[5]={1,2,3,4,5};
int not_specified[]= {5,6,7,64,365,2,2};
int two_dim_array[2][3];
```

You can change the elements in your arrays once they've been declared. However you cannot change the size of the array once declared. You can access an element in you array in the following manner:

```
cout<<one_to_five[3]<<endl; //will return the fourth element, 4
cout<<not_specified[2]<<endl; //will return the third element, 7
cout<<two_dim_array[1][2]<<endl;//will return some random number since
↪  nothing has been declared here and no memory address has been assigned
↪  to this data.
```

You will start running into errors when you try to access elements in array that do not exist, for example:

```
cout<<one_to_five[114]<<endl; //will return some random number or give you a
↪  Segmentation fault (core dumped).
```

In the above case, getting a

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

error message is a blessing because if your calculation or analysis depends on some element of an array, and you make similar typo, all of your results will be wrong (or look correct even though it's not supposed to be) without you knowing why. These are the undefined behaviour of programming. Nothing but good programming habits will help you avoid this kind of problems.

1.3 Vectors

Vector is part of the C++ package that a lot of people love and hate. I am personally on the side of hating it. In order to use C++ vectors, you must first include the vector libraries in your code with:

```
#include <vector>
```

Vectors are similar to arrays except they are declared differently and there are many built-in functions which allow you to increase or decrease the number of element. There are also many safety features not present for the arrays such as out of bounds error.

```
vector<int> v_one_to_five = {1, 2, 3, 4, 5};
cout<<v_one_to_five[3]<<endl; //will return fourth element, 4
cout<<v_one_to_five[114]<<endl; //will return 0.
cout<<v_one_to_five[5]<<endl; //will return 0.
cout<<v_one_to_five[-114]<<endl; //will return 0.
```

Do keep in mind that that for handling large data fast, array is the fastest option in C and C++. You should note that because the arrays are so simple, they are typically compatible with GPU programming (OpenGL and/or NVIDIA CUDA) directly.
The greatest advantage of vector over arrays is that the vectors can change its size after it has been declared in the following manner.

```
v_one_to_five.push_back(6);
```

Using the push_back method, you have now set the v_one_to_five[5] (the sixth element) to 6. There are other built-in ways to manipulate elements in a vector. For more information, refer to the C++ reference at http://www.cplusplus.com/reference/vector/vector/.

2 If statements

The if statement takes in either true or false as its argument. Based on whether if the given argument is true or false, you can determine what the outcome will be. Any and all of the following operators can be used to determine whether if a given argument is true or false.

2.1 logical operators

The logical operators in programming are the following:

```
1 - =
```

equal to.

```
1 - !
```

Not (boolean)

```
1 - &&
```

”and”

```
1 - ||
```

”or”

2.2 Relational and comparison operators

There are also mathematical comparison operators.

```
1 - ==
```

Equal to

```
1 - !=
```

Not equal to

```
1 - <
```

Less than

```
1 - >
```

Greater than

```
1 - <=
```

Less than or equal to

```
1 - >=
```

Greater than or equal to

2.3 Example of an if statement

You should also note that an if statement can have more than one condition. By using logical operators and (&&) and or (||), a single if statement can have infinitely many number of conditions as long as you can can keep track of them.

```
if( (one_to_five[1]=v_one_to_five[1]) && (eight=true) ){
        cout<<"match and true"<<endl;
}
else{
                cout<<"not match or not true"<<endl;
}
if(eight!=true){
        cout<<"false"<<endl;
}
else{
                cout<<"true"<<endl;
}
if(3>=2){
        cout<<"sane"<<endl;
}
    else{
                cout<<"insane"<<endl;
}
```

In the above, you will find that each if statement is followed by an else statement. The else do not need to be specified for all if statement, however it is a good practice to keep them to keep track of your conditions.

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

3 Improving your functions package

1. Modify your functions package from our previous session so that it has the following:
   - A boolean function that returns true or false based on the input float. Anything above 0.5 is true.
   - A boolean function that returns true if the sum of two given integers are higher than 50.
   - A string function that takes in 5 character that returns a string of those 5 characters attached.
   - Make sure that vector headers are loaded and can that vectors can be called.

4 Practicing arrays and vectors

1. Create an array of characters that intakes your first name as characters and outputs it on the screen.
2. Create a vector of characters and do the same as above
3. Create an empty vector of character and use the .push_back function to input your name, and do the same as above.
4. Create a 3x3 integer array and fill it as the following:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

5. Create a 3x3 integer vector and fill it as the following:

$$\begin{bmatrix} 3 & 2 & 3 \\ 1 & 5 & 2 \\ 7 & 9 & 9 \end{bmatrix}$$

6. Create an integer function that can return how many elements in the array and the vector above matches.
7. Create a void function that can perform addition of two 2 x 2 arrays.

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

5 Iterative operations

Up until now, the array and vector operations are tedious as you have been calling each element of array or vector element-by-element. Obviously there are better ways to do this. You can instead tell your computer to look at one element of an array or vector at a time. A proper term for this is to "loop through" an entire array or vector. The technical term for command to "loop through" all of the elements of an array/vector are called "loops". There are two types of loops in C/C++.

- For loop
  For loop is given which element to start with and which element to end at. For standard C for loops you can must define an integer index to loop through all elements in a given array or vector. However, in C++ version, you may also define a type and loop through all elements of a specific type in the array/vector
- While loop
  While loop is given conditions to stop. As long as no stop sign is given, keeps going. While loops are particularly useful for hardware applications and interactive programs since it allows a device or program to remain running until a command is given to turn it off.

5.1 For loop

A standard C for loop uses the following structure:

```
int one_to_five[5]={1,2,3,4,5};
for (int i = 0; i < 5 ; i++){
        cout<<"For index: "<<i<<" element of one_to_five is:
↪  "<<one_to_five[i]<<endl;
}
```

As you may have noticed a for loop is typically declared in the following manner:

```
for (initial_condition; end_condition; action_at_the_end_of_each_iteration){
        //arguments
}
int i =0 //declares a new integer, some index i, starting from 0
i<5 //keep iterating until i = 5
i++ //count "up".
```

Obviously you can count down as well, and use different letter(s) as your index. You can also create a for loop inside a for loop (called nested for loops) for larger single dimension array/vectors. An example of a 2 dimensional for loop can be found below.

```
int two_by_two[2][2];
two_by_two[0][0]=1;
two_by_two[0][1]=2;
two_by_two[1][0]=3;
two_by_two[1][1]=4;
for (int i = 0; i < 2 ; i++){
        for(int j = 0; j < 2 ; j++){
                cout<<"The position (x,y) : ("<<i<<","<<j<<") holds the
↪  value: "<<two_by_two[i][j]<<endl;
        }
}
```

The for loops can be nested infinitely as long as you, the programmer can keep track of each for loop. For more information on the for loops please visit the C++ reference website https://en.cppreference.com/w/cpp/language/for.

5.2 While loop

The while loops are typically easier and simpler to use because you typically only need to give it a single condition. It is also easy to send your computer to "infinite loop" meaning that you've told your computer to do something infinitely many times without a stopping condition. A perfect way to do this can be seen below:

```
while (true){
        cout<<"Keep going until interrupted by Ctrl + C from the command
 ↪  line"<<endl;
}
```

The above is an infinite loop that will keep going until you send the computer to end the program. In many cases if the computer allocates too much resources, the program will keep going until the computer crashes, or if you hard-reset your computer (reboot using power button). For the time being, as long as you are running the code from bash command line, there are several built-in safety features within bash that can help with it, one of them is a key combination of "Control" key and "c" key. This sends an "interrupt signal" to the terminal ending whatever program is running from the terminal.
You can use while loop in a similar way as a for loop as seen below:

```
int i = 0;
while (i<10){
        cout<<"Counting index: "<<i<<endl;
        i++
}
```

The above is equivalent to a for loop:

```
for (int i = 0 ; i < 10 ; i++){
        cout<<"Counting index: "<<i<<endl;
}
```

Another way to stop a while loop is by placing an escape condition. The escape condition will be covered in the next lecture but the following is one example you can inspect for now:

```
int i = 0;
while (true){
        i++
        if(i==10){
                break;
        }
}
```

The if loop that "breaks" the for loop has the escape condition of i==10. For more information visit https://en.cppreference.com/w/cpp/language/while. Let's now try these new tools out.

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

6 Using for loops and while loops on arrays and vectors

1. Using "cin" and "for loop" to create an array characters that intakes your first name as characters and outputs it on the screen using for loop.
2. Create a program that uses "cin" and a single "for loop" to let you enter 9 elements of a 3x3 integer array.
3. Create a program that uses "cin" and 2 nested "for loop" to let you enter 9 elements of a 3x3 integer array.
4. Re-do the previous two items for vectors.
5. Re-do all of the above using "while" loop.
6. Create a program that allows you to build a 4x4 matrix and returns the following:
   (a) Sum of all elements in the matrix.
   (b) Sum of absolute value of all element in the matrix.
   (c) Determinants of the matrix.
   (d) Which element has been entered more than once, and the number of times.
7. Using either for or while loop create a program that allows you to create two 3x3 matrices and allow you to perform:
   (a) Matrix addition of the two matrices.
   (b) Matrix multiplication of the two matrices.
   (c) Diagonalization of each of the 3x3 matrices.

7 Conclusion

Today we have covered primitive data types and simple operations of if statement, for loops and while loops. These are fundamental concepts of all programming languages and they are applicable for everything.
Most important part is that now you are equipped to handle data. You have all of the tools to handle data. In the next lecture we will actually handle some data, if you have not completed all of the exercises today, I strongly recommend you to try them and complete everything at home on your own. Many resources are available online and do not feel afraid to use them. Just do not copy-and-paste. You may literally type in what is written online letter-by-letter but do not "copy and paste".