Universität Zürich[UZH]
Physik-Institut

Lecture 7: Objects and classes

1 Introduction

C++ is an object oriented programming (OOP) language. One of the greatest advantage of OOP is the fact we can have a main code, and cut up everything else into smaller objects.

1.1 Objects and classes

You have already seen objects and classes in this class. Since the beginning of the course, you have been building your functions package. Your functions package has .cxx and an accompanying header file with respective constructors.
The functions package can also have objects embedded in them.
Let's look at the following example starting with the header file for objects, its source code and finally the main

```
1  //myobjects.h
2  #include <iostream> //"including" input/output stream.
3  #include <stdio.h>
4  using namespace std;
5
6
7  class operatingsystem{
8          public:
9          string name;
10         int date_of_publication;
11         void set_name(string newname);
12         string get_name();
13         int age();
14         void set_age(int newage);
15 };
16
17 class car{
18         public:
19         string name;
20         string year;
21         string model;
22         void set_name(string newname);
23         string get_name();
24 };
```

Note that the header file keeps all of the structure of variables. In compound variables as above objects, you can have any number of variables in each object as you wish. Just remember that as you increase the number of inner-variables, size of the object increases as well in terms of memory allocation.
Following is the .cxx file for the accompanying header file.

```cpp
//myobjects.cxx
#include <iostream> //"including" input/output stream.
#include <stdio.h>
#include "myobjects.h"
#include "myfunctions.h"

//Methods for operatingsystem
void operatingsystem::set_name(string newname){
        name=newname;
};

string operatingsystem::get_name(){
        return name;
};

int operatingsystem::age(){
        int result=date_of_publication;
        return result;
};

void operatingsystem::set_age(int newage){
        date_of_publication=newage;
};

//Methods for car

void car::set_name(string newname){
        name=newname;
};

string car::get_name(){
        return name;
};
```

Here, you must define what each of the methods you've defined does. You have to refer back to the header to see what objects use what variables. Finally, let's look at an example main code:

```cpp
//main.cxx
#include<iostream>
#include<stdio.h>
#include<vector>
#include<fstream>
#include<sstream>
#include"myobjects.h"
#include"myfunctions.h"

#include <ctime>

int year(){
        time_t now = time(0);
        tm *lt=localtime(&now);
        int result = 1900+lt->tm_year;
        return result;
```

```
}

int main(int argc,char *argv[]){//Main begins
        cout<<year()<<endl;

        operatingsystem a;
        a.set_name("Windows95");
        cout<<a.get_name()<<endl;
        a.set_name("Linux");
        cout<<a.get_name()<<endl;
        a.set_age(29);
        cout<<a.age()<<endl;

        car c;
        c.set_name("Mercedes");
        cout<<c.get_name()<<endl;

        operatingsystem Linux[5];
        Linux[0].set_name("OpenSUSE");
        Linux[1].set_name("CENTOS7");
        Linux[2].set_name("Ubuntu");
        Linux[3].set_name("Windows 10");
        Linux[4].set_name("Mac OS X");

        for (int i= 0; i<5; i++){
                cout<<Linux[i].get_name()<<endl;
        }
        car * test;
        test=(car*)malloc(20*sizeof(int));
        test->set_name("BMW");
        cout<<test->get_name()<<endl;

    return 0;
}//Main Ends
```

As you may have noticed, similar to functions package, you can have a single objects package with many different objects. The step you need to be aware of is that you need to compile everything together. Suppose you have a myfunctions.h

```
//myfunctions.h
#include <iostream> //"including" input/output stream.
#include <stdio.h>
using namespace std;
void sayhelloto(string input);
```

and accompanying myfunctions.cxx

```
#include <iostream> //"including" input/output stream.
#include <stdio.h>

#include "myobjects.h"
#include "myfunctions.h"

using namespace std;
```

```

void sayhelloto(string input){
        cout<<"Hello "<<input<<endl;
}
```

If you have been compiling your functions package properly in the past, all you need to now is to create an extra step in your Makefile to compile myobject.cxx into myobject.o linker and link all of myobject.o myfunctions.o and main.cxx together. Below you will find an example Makefile you should have made and maintained by now.

```
#This is the directory of YOUR source code.
sourcedirectory=./

#These are your source codes and components
trial=$(wildcard *.cxx)
myobjects=myobjects.cxx
myfunctions=myfunctions.cxx
mymain=main.cxx
output=$(mymain:.cxx=)


#Here, we're defining the compilers.
CC=gcc
CPP=g++
NVCC=nvcc

#We're defining systems variable such as "remove" from system and
↪  "timestamp"
RM=rm
TIMESTAMP=$(shell date +"%Y_%m_%d_T-%H_%M" )

SFLAG=-Wall
PFLAG=-lncurses


objects = $(trial:.cxx=)

#objects =
#When a Makefile is executed, by default it tries the option "all"
all: clean second third
#We will tell the makefile to clean, compile the first component, second
↪  then the third component.


$(objects): %: %.cxx

        @echo Compiling $(sourcedirectory)$<
        @$(CPP) -o $(addprefix run_,$@.exe) $(SFLAG) $< $(PFLAG)
        @echo Successfully compiled $(sourcedirectory)$<
        @echo executable is $(addprefix run_,$@.exe)
second:
#Let's now compile the second part.
        @echo Compiling $(sourcedirectory)$(myobjects)
        @$(CPP) -c -o $(myobjects:.cxx=.o) $(SFLAG)
        ↪  $(sourcedirectory)$(myobjects) $(PFLAG)
```

```
43          @echo Successfully compiled $(sourcedirectory)$(myobjects)
44          @echo The unliked compiled code is $(myobjects:.cxx=.o)
45
46          @echo Compiling $(sourcedirectory)$(myfunctions:.cxx=.o)
47          @$(CPP) -c -o $(myfunctions:.cxx=.o) $(SFLAG)
            ↪  $(sourcedirectory)$(myfunctions) $(PFLAG)
48          @echo Successfully compiled $(sourcedirectory)$(myfunctions)
49          @echo The unliked compiled code is $(myfunctions:.cxx=.o)
50  third:
51  #Let's link the second part
52          @echo Linking $(myobjects:.cxx=.o) and $(myfunctions:.cxx=.o) with
            ↪  $(mymain)
53          @$(CPP) -o $(addprefix run_,$(output).exe) $(SFLAG)
            ↪  $(sourcedirectory)$(myobjects:.cxx=.o)
            ↪  $(sourcedirectory)$(myfunctions:.cxx=.o) $(mymain) $(PFLAG)
54          @echo Successfully compiled $(sourcedirectory)$(output)
55          @echo Everything is linked and compiled into $(addprefix
            ↪  run_,$(output).exe)
56  clean:
57          @echo $(TIMESTAMP)
58          @echo "Making old/$(TIMESTAMP) directory"
59          $(shell mkdir -p old/$(TIMESTAMP) )
60          @echo "Copying the source to the old directory"
61          $(shell cp -r $(sourcedirectory)/*.cxx old/$(TIMESTAMP) )
62          $(shell cp -r $(sourcedirectory)/*.h old/$(TIMESTAMP) )
63          @echo "Moving all .exe to the old directory"
64          $(shell mv *.exe old/$(TIMESTAMP) )
65          $(shell mv *.o old/$(TIMESTAMP) )
66          @echo "Copying the Makefile to the old directory"
67          $(shell cp Makefile old/$(TIMESTAMP) )
```

You should know that there are properties such as inheritance and ”private” variables inside objects but we will not cover that in this course. We will stay ”public”. Now, let’s practice making and using some objects.

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

2 Practice session 1

1. Create the above objects car and operating system as you have seen in the example.
2. Create a method for each object to tell you how old it is.
3. Create a "Getter" and "Setter" for each and all variables for each object.
4. Create an array of operating systems and set their properties for at least 5 operating systems.
5. Create a vector of cars and set their properties for at least 5 cars.
6. Do all of the above using heap memory.
7. Create at least 5 of your own objects with at least 5 properties and 3 methods in each object.

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

3 Questions and catch up time

During this session you may ask questions and try to catch up on everything else that you have not yet completed since the beginning of the course.
In our next class we will move onto a pseudo-programming/scripting using ROOT framework, You should by now be as comfortable as you can with everything that has been covered in the previous lectures. The following practice session will be an exercise to everything that you've learned.

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

4 Practice session 2

1. Start with a new main, makefile, functions package and objects package.
2. Create a random number generator in your functions package.
3. Create a general function that takes in a stack vector and calculates average, max, min and standard deviation.
4. Create a general function that takes in a heap vector and calculates average, median max, min, skew and sample standard deviation.
5. Create a new object called "data" that can hold all of the values above.
6. Modify the data object to store a set of heap or stack array or vector as well.
7. Create an array and vector of data objects in heap, and stack and input 10000 random integers each and return all of their properties (i.e. average, max, min and etc).

5 Conclusion

What you have been doing in the last practical session is essentially the how "ROOT" analysis framework got put together by CERN.
ROOT has a very large number of objects and built-in function with a command line infrastructure which allows you to quickly test and draft parts of your code without compiling.
In the next class, we will go over how we can use the scripting and command line functions in ROOT.
I strongly recommend attempting to install ROOT in your current system. The compilation typically takes in between 1 to 8 hours depending on your systems specification. Therefore we cannot cover it during the class. For more information please visit https://ph-root-2.cern.ch/releases/release-62200/ and https://root-forum.cern.ch/t/wsl-root-install-usr-bin-ldd-line-160-lib-ld-linux-so-40242/13.