CCNU-UOW
CSCI851 Advanced Programming
Autumn 2020

Prof. Zhifeng Wang

Laboratory Exercise 11 (Week 11)

Note that lab exercises marked with a * are effectively extension exercises.

1 Task One: Warm–up exercises

1. Why will the following fail to compile?

```
class B
{
public:
    virtual void X() = 0;
};

class D : B
{
public:
    virtual void X() {cout << "D object" << endl;}
};

int main()
{
    B objB;
}
```

2. Why is the following class not very useful?

```
class X {
  public:
     virtual void x() final = 0;
};
```

3. `Debug-A.cpp` only has one problem, a missing constructor for `CSL`. Fix it!

4. * The file `primes.cpp` contains the code from slide 8 of `S6b`. This code calculates primes at compile time, although not in the best of ways. Time how long compilation takes for a range of values of LAST, which you can set by compiling as follows:

   ```
   $ time CC -DLAST=10 primes.cpp
   ```

   Choose sensible values of LAST to use and continue working on other activities while the program is compiling. Graph the times taken, using something like Excel. A single compilation at a value may not be a good reflection of the typical cost.

2 Task Two: A first function template

Write code `Symbolic.cpp` that contains a function template to display a value preceded and succeeded by n elements of a symbol x on a line, with a space on each side of the value. Write a `main()` function that tests the function with char, int, double and string arguments. The output could be, for example,

```
*** 47 ***
000 39.25 000
aaaa Bob aaaa
```

What would a ADT `X` need in order to work with your function template? Use another appropriate class that you have previously seen to test this.

3 Task Three: Follow the algorithm

Consider the function template and definitions below.

```
template <typename T>
T funcExp(T list[], int size){
        int j;
        T x = list[0];
        T y = list[size-1];
        for(j=1; j<(size-1)/2;j++){
                if (x < list[j]) x = list[j];
                if (y > list[size-1-j]) y = list[size-1-j];}
        return (x+y); }

int list[8]={1,2,9,3,5,8,13,10};
string strlist[]={"one","fish","two","fish","red","fish","blue","fish"};
```

Determine the output of the following statement, firstly without implementing the code and then check your solution by implementing it in `funcExp.cpp`.

```
cout << funcExp(list,8) << " :: " << funcExp(strlist,8) << endl;
```

4 Task Four: A class template

Write a class template `Two<R,S>` with the two parameterised types `R` and `S`. Write the class template and a `main()` function with appropriate code to test the following functionality as you add it.

1. The class should have a constructor that takes one object of each type and uses those to set the values in the object.
2. There should be a display function that outputs the object values.
3. There should be a function that attempts to add the two variables together.

What happens if you attempt to instantiate a template class based on this class template for types where at least one of the operations doesn't make sense.

5 * Task Five: A variadic function

Write a recursive variadic function to print out the number of arguments in the pack being passed in each call to a display function, along with the argument being outputted at the time. This is a minor modification of the code in the lecture notes set `S6b`.

6 * Task Six: A little bit more

The file `Bits-of-Memory` contains three versions of the same class. Data elements that correspond to Boolean states (0 or 1) are stored in three different ways. Firstly using an `int` to store each data element, secondly using a `bool` to store each, and thirdly using a `single bit` to store each data element.

The advantage of using `single bit` fields is that of saving space. We can, more generally, use `unsigned x:y`, where y is a positive integer, to indicate that we associate y bits with the variable x. You can also do things like this ...

```
unsigned char b1 : 3, : 2, b2 : 6, b3 : 2;
```

to leave some of the bits empty.

There is more information at:

`http://en.cppreference.com/w/cpp/language/bit_field`

1. Explore the use of this.
2. How large does `y` need to be for one of the flags in `DriverUsingBits` before the size of the objects of the class increases?
3. How large can `y` usefully be?
4. On Banshee each file has a permission string listed with it, tied to what users can do. So ...

   ```
   $ ls -a
   -rw-------   1 lukemc   csstf       1336 Sep 18 13:13 Bits-of-Memory.cpp
   -rw-rwxrw-   1 lukemc   csstf       1340 Sep 18 12:53 Dog.cpp
   ...
   ```

   The permission string is the `-rw-------` at the start. Ignoring the 1st dash, the next 3 correspond to (r)ead, (w)rite, and e(x)ecute permissions for the owner (lukemc), the next 3 for the group (csstf), and the last for everybody else. So in the listing above the owner can read and write `Dog.cpp`, the group can read, write, and execute `Dog.cpp`, and everybody else can read and write `Dog.cpp`.

   Write a bit based class to store, and interact with, permission strings.