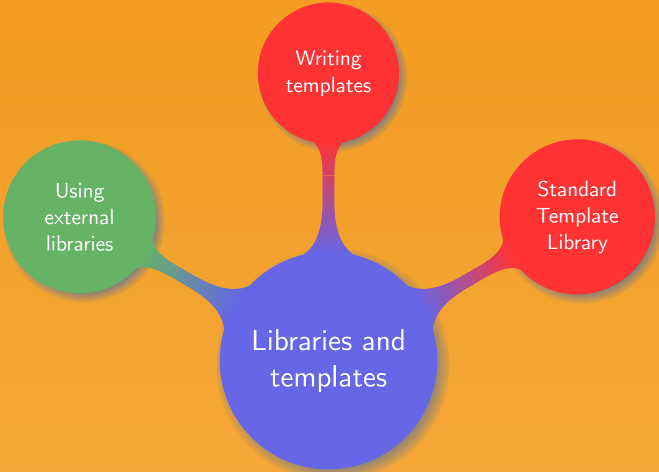
Introduction to Computer and Programming

11. Libraries and templates

Manuel – Summer 2019

Chapter organisation



Writing templates

Using external libraries

Standard Template Library

Libraries and templates

Libraries

Simple overview:

- Many libraries available to define all type of objects
- Using a library:
  - Include header files
  - Possibility to use the library namespace
  - Reference the library at compilation time

Libraries

Simple overview:

- Many libraries available to define all type of objects
- Using a library:
  - Include header files
  - Possibility to use the library namespace
  - Reference the library at compilation time

To use a library the compiler must know:

- Where the header files are located
- The namespace a function belongs to
- Where the machine code is located

The OpenGL library

Overview:

- Open Graphic Library (openGL)
- C library for drawing
- Cross platform
- Multi platform Application Programming Interface (API)
- API interacts with the GPU
- Widely used in games, Computer Aided Design (CAD), flight simulators, etc.

The OpenGL library

Overview:

- Open Graphic Library (openGL)
- C library for drawing
- Cross platform
- Multi platform Application Programming Interface (API)
- API interacts with the GPU
- Widely used in games, Computer Aided Design (CAD), flight simulators, etc.

Our goal is to wrap the C functions into classes and build a home

Hierarchy diagram

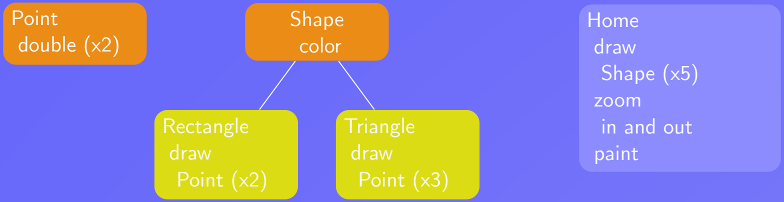
First steps:

- Identify all the objects
- Organise them using a hierarchy diagram
- Identify the methods
- Define the necessary attributes

Hierarchy diagram

First steps:

- Identify all the objects
- Organise them using a hierarchy diagram
- Identify the methods
- Define the necessary attributes



Point
double (x2)

Shape
color

Rectangle
draw
Point (x2)

Triangle
draw
Point (x3)

Home
draw
Shape (x5)
zoom
in and out
paint

Figures specification

home/figures.h

```cpp
#ifndef __FIGURES_H__
#define __FIGURES_H__
typedef struct _Point { double x,y; } Point;
class Shape {
  public: virtual void draw() = 0; virtual ~Shape();
  protected: float r, g, b;
};
class Rectangle : public Shape {
  public: Rectangle(Point pt1={-.5,-.5}, Point pt2={.5,.5},
            float r=0, float g=0, float b=0);
    void draw();
  private: Point p1,p2;
};
class Triangle : public Shape {
  public: Triangle(Point pt1={-.5,-.5}, Point pt2={.5,-.5},
            Point pt3={0,.5}, float r=0, float g=0, float b=0);
    void draw();
  private:  Point p1,p2,p3;
};
#endif
```

Figures implementation

home/figures.cpp

```cpp
#include <GL/glut.h>
#include "figures.h"
Shape::~Shape(){}
Rectangle::Rectangle(Point pt1, Point pt2,
    float red, float green, float blue) {
  p1=pt1; p2=pt2; r=red; g=green; b=blue;
}
void Rectangle::draw() {
  glColor3f(r, g, b); glBegin(GL_QUADS);
  glVertex2f(p1.x, p1.y); glVertex2f(p2.x, p1.y);
  glVertex2f(p2.x, p2.y); glVertex2f(p1.x, p2.y); glEnd();
}
Triangle::Triangle(Point pt1, Point pt2, Point pt3,
    float red, float green, float blue) {
  p1=pt1; p2=pt2; p3=pt3; r=red; g=green; b=blue;
}
void Triangle::draw() {
  glColor3f(r, g, b); glBegin(GL_TRIANGLE_STRIP);
  glVertex2f(p1.x, p1.y); glVertex2f(p2.x, p2.y); glVertex2f(p3.x, p3.y);
  glEnd();
}
```

Home specification

home/home.h

```
#ifndef __HOME_H__
#define __HOME_H__
#include "figures.h"
class Home {
  public:
    Home(Point pt1={0,-.25}, double width=1,
        double height=1.3, double owidth=.175);
    ~Home();
    void draw();
    void zoom(double *width,double *height,double *owidth);
  private:
    Point p; double w, h, o; Shape *sh[5];
    void zoomout(double *width,double *height,double *owidth);
    void zoomin(double *width,double *height,double *owidth);
    void paint(float *r, float *g, float *b);
};
#endif
```

Home implementation (part 1)

home/home_part1.cpp

```cpp
#include <ctime>
#include <cstdlib>
#include "home.h"
Home::Home(Point pt1, double width, double height, double owidth){
  float r, g, b; Point p1, p2, p3;
  p=pt1; w=width; h=height; o=owidth; srand(time(0));
  p1={p.x-w/2,p.y-w/2}; p2={p.x+w/2,p.y+w/2};
  paint(&r,&g,&b); sh[0]=new Rectangle(p1,p2,r,g,b);
  p1={p.x-o,p.y-w/2}; p2={p.x+o,p.y};
  paint(&r,&g,&b); sh[1]=new Rectangle(p1,p2,r,g,b);
  p1={p.x-2*o,p.y+o}; p2={p.x-o,p.y+2*o};
  paint(&r,&g,&b); sh[2]=new Rectangle(p1,p2,r,g,b);
  p1={p.x+w/2-2*o,p.y+o}; p2={p.x+w/2-o,p.y+2*o};
  paint(&r,&g,&b); sh[3]=new Rectangle(p1,p2,r,g,b);
  p1={p.x,p.y+h-w/2}; p2={p.x-w/2,p.y+w/2}; p3={p.x+w/2,p.y+w/2};
  paint(&r,&g,&b); sh[4]=new Triangle(p1,p2,p3,r,g,b);
}
Home::~Home(){ for(int i=0;i<5;i++) delete sh[i]; }
```

Home implementation (part 2)

home/home_part2.cpp

```cpp
void Home::draw() {for(int i=0;i<5;i++) sh[i]->draw();}
void Home::zoom(double *width, double *height, double *owidth){
  int static i=0;
  if(h>=0.1 && i==0) zoomout(width, height, owidth);
  else if (h<=2) { i=1; zoomin(width, height, owidth); }
  else i=0;
}
void Home::zoomout(double *width, double *height, double *owidth){
  h/=1.01; *height=h; w/=1.01; *width=w; o/=1.01; *owidth=o;
}
void Home::zoomin(double *width, double *height, double *owidth){
  h*=1.01; *height=h; w*=1.01; *width=w; o*=1.01; *owidth=o;
}
void Home::paint(float *r, float *g, float *b) {
  *r=(float)rand()/RAND_MAX; *g=(float)rand()/RAND_MAX;
  *b=(float)rand()/RAND_MAX;
}
```

Home instantiation

home/main.cpp

```cpp
#include <GL/glut.h>
#include "home.h"
void TimeStep(int n) {
  glutTimerFunc(n, TimeStep, n); glutPostRedisplay();
}
void glDraw() {
  double static width=1, height=1.5, owidth=.175;
  Home zh({0,-.25},width,height,owidth);
  zh.zoom(&width, &height, &owidth);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  zh.draw(); glutSwapBuffers(); glFlush();
}
int main (int argc, char *argv[]) {
  glutInit(&argc, argv);
  //  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
  glutCreateWindow("Home sweet home");
  glClearColor(1.0, 1.0, 1.0, 0.0); glClear(GL_COLOR_BUFFER_BIT);
  glutDisplayFunc(glDraw); glutTimerFunc(25, TimeStep, 25);
  glutMainLoop();
}
```

Basics

Basic process when using OpenGL:

1. Initialise the library: `glutInit(&argc, argv);`
2. Initialise the display: `glutInitDisplay(GLUT_RGB|GLUT_SINGLE);`
3. Create window: `glutCreateWindow(windowname);`
4. Set the clear color: `glClearColor(r,g,b);`$(r, g, b \in [0, 1])$
5. Clear the screen: `glClear(GL_COLOR_BUFFER_BIT);`
6. Register display callback function: `glutDisplayFunc(drawfct);`
7. Redraw the screen: recursive call to a timer function
8. Start the loop: `glutMainLoop();`
9. Draw the objects

Remarks

Understanding the code:

- Why is the `static` keyword used in both the `glDraw` and `zoom` functions?
- Why were pointers used in he `zoom, zoomin` and `zoomout` functions?
- How were inheritance and polymorphism used?
- Comment the choices of `public` or `private` attributes and methods
- How is the keyword `#ifndef` used?

Compilation

Compiling and running the home:

```
sh $ g++ -std=c++11 -o home main.cpp home.cpp\ figures.cpp
    -lglut -lGL
sh $ ./home
```

Compilation

Compiling and running the home:

```
sh $ g++ -std=c++11 -o home main.cpp home.cpp\ figures.cpp
    -lglut -lGL
sh $ ./home
```

Better strategy is to use a `Makefile`:

- Simple text file explaining how to compile a program
- Useful for complex programs
- Easily handles libraries and compiler options

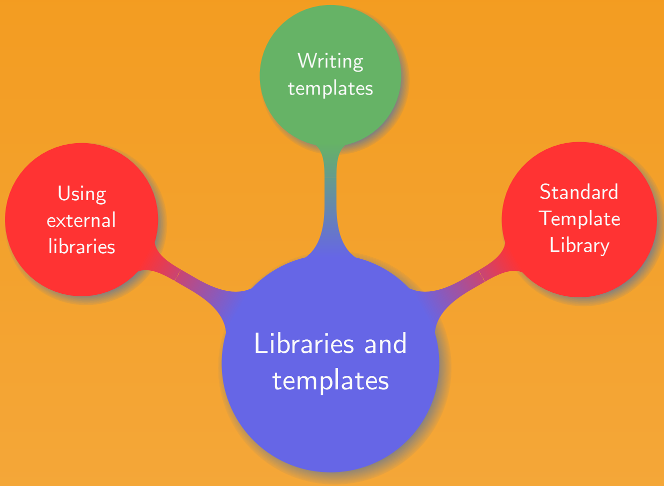
```
sh $ make
```

Makefile

home/Makefile

```makefile
CC = g++ # compiler
CFLAGS = -std=c++11 # compiler options
LIBS = -lglut -lGL # libraries to use
SRCS = main.cpp home.cpp figures.cpp
MAIN = home
OBJS = $(SRCS:.cpp=.o)
.PHONY: clean # target not corresponding to real files
all:    $(MAIN) # target all constructs the home
  @echo Home successfully constructed
$(MAIN):
  $(CC) $(CFLAGS) -o $(MAIN) $(SRCS) $(LIBS)
.cpp.o: # for each .cpp build a corresponding .o file
  $(CC) $(CFLAGS) -c $<  -o $@
clean:
  $(RM) *.o *~ $(MAIN)
```

Chapter organisation



Classes

Limitations of inheritance and polymorphism:

- High level classes, e.g. boat, company, car, etc.
- Low level classes used to define high level ones
- Still need to use function overloading to apply a function to more than one data type

This results in duplicated code, and programs harder to debug

Defining a template

A templates *is a “special class” where the data type is a parameter*

Example.

complex.h

```
#include <iostream>
using namespace std;
template<class TYPE>
class Complex {
  public:
    Complex(){ R = I = (TYPE)0; }
    Complex(TYPE real, TYPE img) {R=real;I=img;}
    void PrintComplex() {cout<<R<<'+'<<I<<"i\n";}
  private:
    TYPE R, I;
};
```

Using a template

To use a template add the data type to the class name:

```
complex<float> c1; complex<int> c2;
typedef complex<double> dcplx; dcplx c3;
```

Exercise.

Using the previous complex template, display `Complex` numbers composed of the types: `int`, `double` and `char`

Using a template

To use a template add the data type to the class name:

```cpp
complex<float> c1; complex<int> c2;
typedef complex<double> dcplx; dcplx c3;
```

Exercise.

Using the previous complex template, display `Complex` numbers composed of the types: `int`, `double` and `char`

complex.cpp

```cpp
#include "complex.h"
typedef Complex<char> CComplex ;
int main () {
  Complex<double> a(3.123,4.9876); a.PrintComplex();
  Complex<int> b; b = Complex<int>(3,4);
  b.PrintComplex();
  CComplex c('a','b'); c.PrintComplex();
}
```

A bit of history

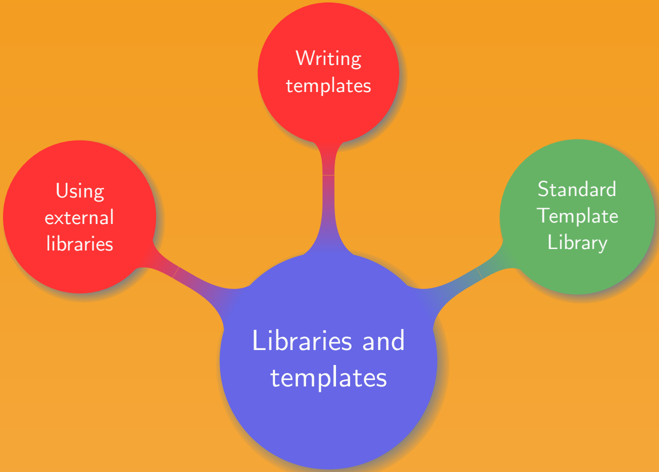
A few dates:

- 1983: C++
- 1994: templates accepted in C++
- 2011: many fixes/improvements on templates

Notes on templates:

- They are very powerful, complex and new
- They are not always handled nicely
- They might lead to long and unclear error messages
- They are not always fully optimized
- They require much work from the compiler

Chapter organisation



Libraries and templates

Writing templates

Using external libraries

Standard Template Library

Basics on STL

C++ is shipped with a set of templates:

- Standard Template Library (STL)
- STL goals: abstractness, generic programming, no loss of efficiency
- Basic idea: use templates to achieve compile time polymorphism
- Components:
  - Containers
  - Iterators
  - Algorithms
  - Functional

Sequence containers

Common sequence containers:

- Vector: automatically resizes, fast to access any element and to add/remove elements at the end
- Deque: vector with reasonably fast insertion deletion at beginning and end, potential issues with the iterator
- List: slow lookup, once found very fast to add/remove elements

Sequence containers

Common sequence containers:

- Vector: automatically resizes, fast to access any element and to add/remove elements at the end
- Deque: vector with reasonably fast insertion deletion at beginning and end, potential issues with the iterator
- List: slow lookup, once found very fast to add/remove elements

A few other available containers:

- Set
- Multiset
- Multimap
- Bitset
- Valarray

Vectors

A vector is similar to an array whose size can be changed:

- Size: automatically adjusted
- Template: no specific initial type
- A few useful functions: `push_back`, `pop_back`, `swap`

Example.

```
#include <vector>
vector<int> vint;
vector<float> vfloat;
```

Vectors

vect.cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main () {
  vector<int> v1(4,100); vector<int> v2;
  vector<int>::iterator it;
  v1[3]=5;
  cout << v1[3] << " " << v1[0] << endl;
  v2.push_back(2); v2.push_back(8); v2.push_back(18);
  cout << v2[0] << " " << v2[1] << " " << v2[2] << endl;
  v2.swap(v1);
  cout << v2[1] << " " << v1[1] << " " << v1.size() << endl;
  v1.erase(v1.begin()+1,v1.begin()+3);
  cout << v1[0] << " " << v1[1] << " " << v1.size() << endl;
  v1.pop_back();
  cout << v1[0] << " " << v1[1] << " " << v1.size() << endl;
  for(it=v2.begin(); it!=v2.end();it++) cout << *it << endl;
}
```

Container adaptors

Common containers adaptors:

- Queue: First In First Out (FIFO) queue → list, deque
  Main methods: `size`, `front`/`back` (access next/last element), `push` (insert element) and `pop` (remove next element)
- Priority queue: elements must support comparison (determining priority) → vector, deque
- Stack: Last In First Out (LIFO) stack → vector, list, deque
  Main methods: `size`, `top` (access next element), `push` and `pop` (remove top element)

Queue

queue.cpp

```cpp
#include <iostream>
#include <queue>
using namespace std;
int main () {
  int i,j=0;
  queue <int> line;
  for(i=0;i<200;i++) line.push (i+1);
  while(line.empty() == 0) {
    cout << line.size () << " persons in the line\n"
      << "first in the line: " << line.front() << endl
      << "last in the line: " << line.back() << endl;
    line.pop ();
    if(j++%3==0) {
      line.push (++i);
      cout << "new in the line: " << line.back() <<endl;
    }
  }
}
```

Iterators

A new object:

- Object that can iterate over a container class
- Iterators are pointing to elements in a range
- Their use is independent from the implementation of the container class

```
for(i=0;i<vct.size();i++) {
  ...
}
```

```
for(it=vct.begin(); \
 it !=vct.end();++it) {
  ...
}
```

Efficiency of `vct.size()`: fast operation for vectors, slow for lists

Iterators

Example.

iterator.cpp

```cpp
#include <iostream>
#include <set>
using namespace std;
int main() {
  set<int> s;
  s.insert(7);s.insert(2);s.insert(-6);
  s.insert(8);s.insert(1);s.insert(-4);
  set<int>::const_iterator it;
  for(it = s.begin(); it != s.end(); ++it)  {
    cout << *it << " ";
  }
  cout << endl;
}
```

Algorithms templates

Common algorithms implemented in templates:

- Manipulate data stored in the containers
- Mainly targeting range of elements
- Many “high low-level” functions such as:
  - Sort
  - Shuffle
  - Find with conditions
  - Partition

count

In a given range returns how many element are equal to some value

Example.

count.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
int main () {
  string colors[8] = {"red","blue","yellow","black",
    "green","red","green","red"};
  vector<string> colorvect(colors, colors+8);
  int  nbcolors = count (colorvect.begin(),
      colorvect.end(), "red");
  cout << "red appears " << nbcolors  << " times.\n";
}
```

find

In a given range, returns an iterator to the first element that is equal to some value, or the last element in the range if no match is found

Example.

find.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
int main () {
  string colors[8] = {"red","blue","yellow","black",
    "green","red","green","red"};
  vector<string> colorvect(colors, colors+8);
  vector<string>::iterator it;
  it=find(colorvect.begin(), colorvect.end(), "blue"); ++it;
  cout << "following blue is " << *it << endl;
}
```

unique

Remove consecutive duplicates

Example.

unique1.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
bool cmp(string s1, string s2) { return(s1.compare(s2)==0);}
int main () {
  string colors[8] = {"red","blue","yellow","black",
    "green","green","red","red"};
  vector<string> colorvect(colors, colors+8);
  vector<string>::iterator it;
  it=unique(colorvect.begin(), colorvect.end(),cmp);
  colorvect.resize(distance(colorvect.begin(),it));
  for(it=colorvect.begin(); it!=colorvect.end();++it)
    cout << ' ' << *it;
  cout << endl;
}
```

sort

Sort elements in ascending order

Example.

sort.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
bool cmp(string s1, string s2) { return(s1.compare(s2)<0);}
int main () {
  string colors[8] = {"red","blue","yellow","black",
    "green","green","red","red"};
  vector<string> colorvect(colors, colors+8);
  vector<string>::iterator it;
  sort(colorvect.begin(), colorvect.end(),cmp);
  for(it=colorvect.begin(); it!=colorvect.end();++it)
    cout << ' ' << *it;
  cout << endl;
}
```

Removing all duplicates

Exercise.

Remove all duplicate elements from the color vector.

Removing all duplicates

Exercise.

Remove all duplicate elements from the color vector.

unique2.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
bool cmp1(string s1, string s2) {return(s1.compare(s2)<0);}
bool cmp2(string s1, string s2) {return(s1.compare(s2)==0);}
int main () {
  string colors[8]={"red","blue","yellow","black","green","green","red","red"};
  vector<string> colorvect(colors, colors+8); vector<string>::iterator it;
  sort(colorvect.begin(), colorvect.end(),cmp1);
  it=unique(colorvect.begin(), colorvect.end(),cmp2);
  colorvect.resize(distance(colorvect.begin(),it));
  for(it=colorvect.begin(); it!=colorvect.end();++it)  cout << ' ' << *it;
  cout << endl;
}
```

reverse

Reverse the order of the elements

Example.

reverse.cpp

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
int main () {
  string colors[8] = {"red","blue","yellow","black",
    "green","green","red","red"};
  vector<string> colorvect(colors, colors+8);
  vector<string>::iterator it;
  reverse(colorvect.begin(), colorvect.end());
  for(it=colorvect.begin(); it!=colorvect.end();++it)
    cout << ' ' << *it;
  cout << endl;
}
```

remove_if

Remove elements and returns an iterator to the new end

Example.

remove.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
bool bstart(string s) { return(s[0]!='b'); }
int main () {
  string colors[8] = {"red","blue","yellow","black",
    "green","green","red","red"};
  vector<string> colorvect(colors, colors+8);
  vector<string>::iterator it;
  it=remove_if(colorvect.begin(),colorvect.end(),bstart);
  colorvect.resize(distance(colorvect.begin(),it));
  for(it=colorvect.begin(); it!=colorvect.end();++it)
    cout << ' ' << *it;
  cout << endl;
}
```

random_shuffle

Randomly rearrange elements

Example.

random.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
int main () {
  srand (unsigned(time(0)));
  string colors[8] = {"red","blue","yellow","black",
    "green","green","red","red"};
  vector<string> colorvect(colors, colors+8);
  vector<string>::iterator it;
  random_shuffle(colorvect.begin(),colorvect.end());
  for(it=colorvect.begin(); it!=colorvect.end();++it)
    cout << ' ' << *it;
  cout << endl;
}
```

minmax

Returns min and max of two elements or the min and max in a list

minmax.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
using namespace std;
bool cmp(string s1, string s2) {return(s1.compare(s2)<0);}
int main () {
  srand (unsigned(time(0)));
  auto mm=minmax({"red","blue","yellow","black"},cmp);
  cout <<  mm.first << ' ' << mm.second;
  cout << endl;
}
```

Key points

- How to use external libraries?
- How to write a Makefile?
- What is the Standard Template Library?
- Why using STL?

Thank you!