ENGR101 T1 2020

Engineering Technology

Arthur Roberts

School of Engineering and Computer Science
Victoria University of Wellington

How to manage the big code?

As we go from one project to another - more and more code is needed to complete the task.
Reminder:

- Project1 - make functions to calculate frequency and amplitude.
- Project2 - use **structs** to make code shorter. Example: bounding rectangle, pixel as struct

Project 3 - even more code.
Code for this project can take from 200 lines, if program is well designed. I seen it done in 100 lines - but it was an outlier.
Program can be up to 1000 lines if it is copy-paste job.
How to make program cleaner and easier to work with?
Manage the complexity of it - introduce your own data types, fit for the problem, make your own functions to work for them.
Then you don't have to deal with **int**, **arrays** and likes. You are working on higher level.
One way to achieve that is to use **class**es.

Classes

- Classes are step up from **struct**s.
- struct allows you to group variables together, creating container for the variables.
- Class allows you to add functions to this container (and a lot of other things too).

Class example

Listing 1: Pixel as a class

```
#include <iostream>

class Pixel{
    unsigned char red,green,blue;
  public:
 void printPixel(){
   std::cout<<"r="<<(int)red<<"␣g="<<(int)green;
   std::cout<<"␣b="<<(int)blue<<std::endl;
  }
  void set(unsigned char r,unsigned char g,unsigned char b){
   red = r; green = g; blue =b;
  }
};

int main(){
  Pixel pix;
  pix.set(10,34,234);
  pix.printPixel();
}
```

Access specifiers

- Public: The public members of a class can be accessed from anywhere in the program using the direct member access operator (.) with the object of that class.
- Private: The class members declared as private can be accessed only by the functions inside the class. Private functions can only be called inside functions of same class.
- Protected: miss it for now

How to get variables inside the class?

If program needs value of the variable inside the class - make function to return it.

Listing 2: Caption

```
class Pixel{
  private:
    unsigned char red,green,blue;
  public:
    void printPixel(){
      std::cout<<"r="<<(int)red<<" g="<<(int)green;
      std::cout<<" b="<<(int)blue<<std::endl;
    }
    void set(unsigned char r,unsigned char g,unsigned char b){
      red = r; green = g; blue =b;
    }
    unsigned char getRed(){
      return red;
    }
};
```

File with class implementation will get too big

So far everything went between opening and closing bracket of the class. If class is big - you need to scroll a lot. There is a better way - leave function **declarations** inside the class but move function **implementation**s outside. When implementation is moved out - compiler should know which class we are talking about: different classes can have functions with same name.

Listing 3: Implementations out of class body

```
class Pixel{
  private:
    unsigned char red,green,blue;
  public:
    void printPixel(); //declarations
    void set(unsigned char r,unsigned char g,unsigned char b);
    unsigned char getRed();
};
void Pixel::printPixel(){
  std::cout<<"r="<<(int)red<<" g="<<(int)green;
  std::cout<<" b="<<(int)blue<<std::endl;
}
void Pixel::set(unsigned char r,unsigned char g,unsigned char b){
  red = r; green = g; blue =b;
}
unsigned char Pixel::getRed(){
  return red;
}
```

This mysterious ::

That should explain this **::** which we seen in

Listing 4: Caption

```
std::cout<<"Hello"<<std::endl;
```

:: is **scope resolution operator**. It is possible to have several **cout**s (one for writing to the screen, one for writing to the network socket, yet another for writing to the file) as long **namespace** (in this case **std**, for standard) is different.

Move class in different file altogether

And it gets better. For now we still have to scroll a lot - everything is in one file.
Whole code for the **Pixel** class can be moved into different file (say, **pixel.hpp**).
If we do that we need to add:

Listing 5: Caption

```
#include "pixel.hpp"
```

to our main file.
Remember **"** indicates that hpp file is in the same folder.

Can we build class using another class inside?

Listing 6: Yes we can

```
#include "pixel.hpp"
class Image{
  private:
    Pixel pixels[10];
  public:
    void setGray();
    void printAllPixels();
};
void Image::setGray(){
  for (int i =0 ; i< 10 ; i++) { pixels[i].set(25,25,25); }
}

void Image::printAllPixels(){
  for (int i = 0 ; i< 10 ; i++) { pixels[i].printPixel(); }
}

int main(){
  Image image;
  image.setGray();
  image.printAllPixels();
}
```

Not all functions are equal - constructor

Listing 7: Caption

```
#include <iostream>

class Pixel{
private:
  unsigned char red,green,blue;
public:
  Pixel();
  void printPixel(); //declarations
  void set(unsigned char r,unsigned char g,unsigned char b);
  unsigned char getRed();
};

Pixel::Pixel(){ // constructor
  std::cout<<"␣Inside␣constructor"<<std::endl;
}

void Pixel::printPixel(){
  std::cout<<"r="<<(int)red<<"␣g="<<(int)green;
  std::cout<<"␣b="<<(int)blue<<std::endl;
}
void Pixel::set(unsigned char r,unsigned char g,unsigned char b){
  red = r; green = g; blue =b;
}
```

Constructor is called every time variable of type Class (called Object) is created. It is inserted into your program automatically in newer versions of C++.
If you want to perform specific action (like reserving memory for the array) on creation of the Object to be done - implement the constructor.
Constructor should have same name as Class. It does not return anything.

What are benefits of Object Oriented Programming(OOP)?

- Structure: It enforce rules on a programmer that, in the long run, help to get more work done
- Encapsulation: Program is broken into pieces and pieces are self-sufficient. They can be tested and debugged separately.
- Classes can be re-used. In simplest case it can be copy-paste. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch.

How to use it for this Project?

There are two distinct parts in the project:

- Image processing
- Robot movement

What image processing part should do?

- Scan the line, detect if there are white/red pixels on this line
- Or there can no white/red pixels anywhere - robot lost
- Lines can be horizontal or vertical

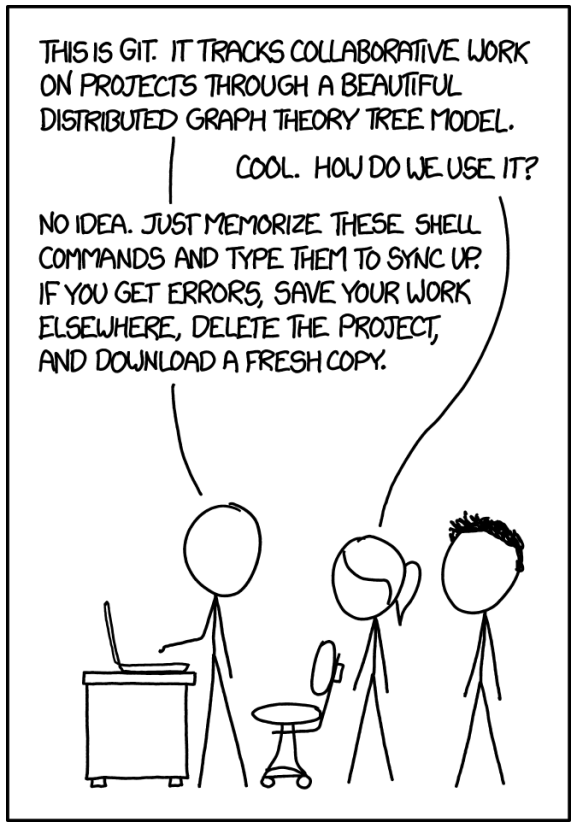
It will pay to make one class for the robot (movement functions go there) and inside make another class for image processing.
Have a look at **robot.hpp** file provided. It is OK to restructure it to your liking.

It is group project

- Teams of four. Selection somewhat random.
- Always source of big drama before.
- First time done remotely.
- Use any communication channel you are comfortable with: Facebook, Discord, e-mail, Microsoft Teams, Zoom...
- Code sharing - GitHub please. And we want to see your repository.

Team code sharing - GitHub

https://git-scm.com/book/en/v2/Getting-Started-Git-Basics



https://xkcd.com/1597/

Team code sharing - GitHub

- GitHub is code cloud storage platform which allows collaboration when writing code, including version control.
- Why not use DropBox or Google Drive? GitHub allows several developer to work on same piece of code without creating "races" - whoever overwrites file on the server first.
- You will use GitHub essentials like repositories, branches, commits, and Pull Requests.

To start:
Go to `https://github.com/`
Register and create repository.

Create new repository

- Repository is where you store all project files
- Give your repository the name you want and the description. Choose ‘Public’ and check the Initialize this repository with a README.md file.
- A README.md is a short text document that introduces the rest of the files and code within a repository. It will often include installation and usage instructions, as well as a To Do list of what needs to be fixed or improved on in the repo.
- Finally, select the green ‘Create Repository’ button.
- To add other members of the team go to **settings**, **Manage Access** and add members of your team (Collaborators).

Uploading files

- A simple way of uploading multiple files however is to use the ‘Upload files’ button where you can simply drag and drop existing files and upload them.
- You will be prompted for a comment on what you just did (the default for you will say ‘Create new file’) at the bottom of the web page. This is known as a commit message and allows you to keep track of the files.
- Then can select the green ‘Commit new file’ button.

Branches

- Branching is the way to work on different versions of a repository at one time.
- By default your repository has one branch named master which is considered to be the definitive branch. We use branches to experiment and make edits before committing them to master.
- When you create a branch off the master branch, you're making a copy, or snapshot, of master as it was at that point in time. If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.

Commit

Inside new branch:

- Make and commit changes
- Edit one of the files: Click the pencil icon in the upper right corner of the file view to edit.
- Write a commit message that describes your changes. Your individual log marks will be based on quality of these commit messages.
- Click Commit changes button.

Pull request

Pull request tell the changes done in the file. It requires other contributors to view it as well as merge it with the master branch.

- Click the ‘Pull requests’ tab.
- Click ‘New pull request’, select the branch and click changed file to view changes between the two files present in our repository.
- Click “Create pull request”.
- Enter any title, description to your changes and click on “Create pull request”.

Merge

Here comes the last command which **merge** the changes into the main master branch.

- Click on “Merge pull request” to merge the changes into master branch.
- Click “Confirm merge”.
- You can delete the branch once all the changes have been incorporated and if there are no conflicts.