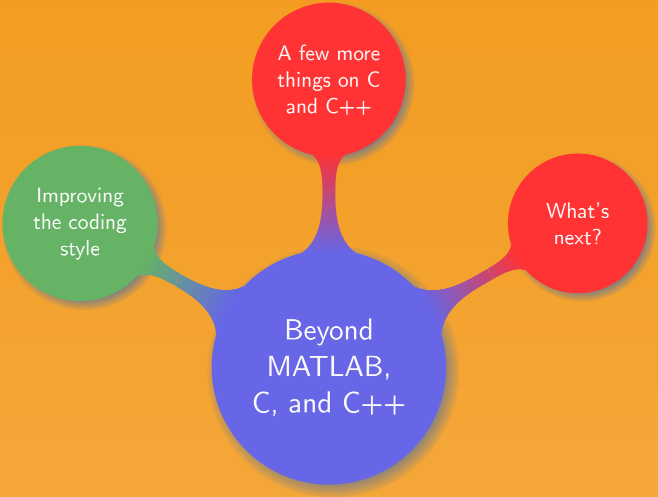
Introduction to Computer and Programming

12. Beyond MATLAB, C, and C++

Manuel – Summer 2019

Chapter organisation



Beyond MATLAB, C, and C++

Improving the coding style

A few more things on C and C++

What's next?

Layer programming

Clean coding strategy:

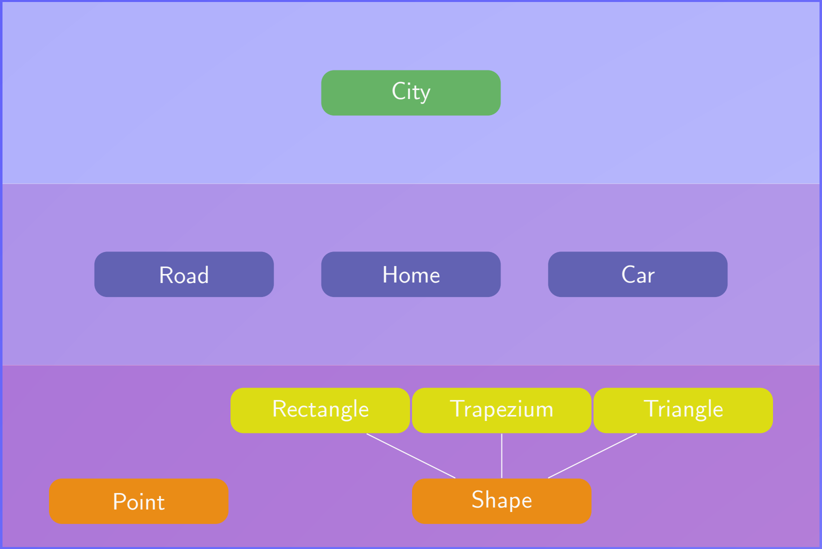
- Split the code into functions
- Organise the functions in different files
- Functions are organised by layers
- Functions of lower layers do not call functions of higher layers
- A function can only call functions of same or lower levels

Layer programming

Example.
In the implementation of the home:

- Lowest layer: definition of the figures (points, rectangle, and triangle)
- Middle layer: definition of the home (home and actions on the home)
- Top layer: instantiation of the home (more actions such as construction of a compound)

Layer programming



Makefile

Makefile

```
CCC = g++
CCFLAGS = -std=c++11 -Wall -Wextra -Werror -pedantic
LIBS = -lglut -lGL
LLIBS = -L. -lhome -lfig
LFIG_SRC = figures.cpp
LFIG_OBJ = $(LFIG_SRC:.cpp=.o)
LFIG = libfig.a
LHOME_SRC = home.cpp
LHOME_OBJ = $(LHOME_SRC:.cpp=.o)
LHOME = libhome.a
MAIN_SRC = main.cpp
MAIN = home
.PHONY: clean hlibs

all: $(LFIG_OBJ) $(LHOME_OBJ) hlibs $(MAIN)
  @echo Home successfully constructed

$(MAIN): $(MAIN_SRC)
  $(CCC) $(CCFLAGS) -o $(MAIN) $(MAIN_SRC) $(LIBS) $(LLIBS)

.cpp.o:
  $(CCC) $(CCFLAGS) -c $< -o $@

hlibs :
  ar rcs $(LFIG) $(LFIG_OBJ);   ar rcs $(LHOME) $(LHOME_OBJ)

clean:
  $(RM) *.o *.a *~ $(MAIN)
```

More compilation

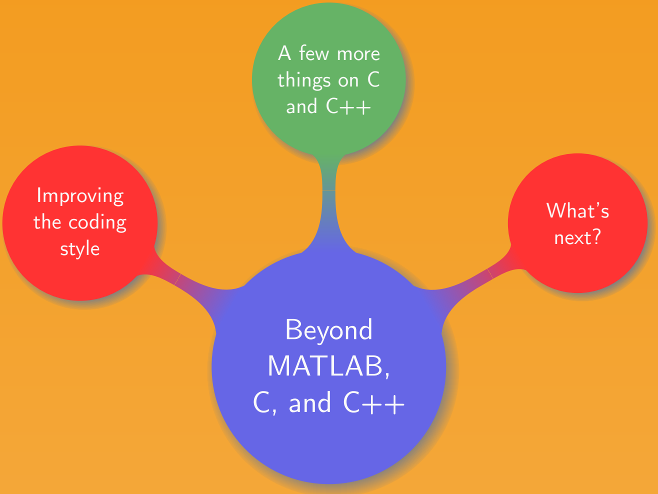
Clean code respecting standards

```
sh $ gcc -Wall -Wextra -Werror -pedantic file.c
sh $ g++ -Wall -Wextra -Werror -pedantic file.cpp
```

When coding:

- Ensure compatibility over various platforms
- Use tools such as *valgrind* to assess the quality of the code (e.g. spot memory leaks)
- For more complex program use a debugger such as *gdb*

Chapter organisation



Beyond MATLAB, C, and C++

A few more things on C and C++

Improving the coding style

What's next?

The `const` keyword

Constant variable:

- Creates a read-only variable
- Use and abuse `const` if a variable is not supposed to be modified
- In the case of a `const vector` use a `const` iterator:

```
vector<T>::const_iterator
```

Constant pointers vs. pointer to constant

Constant pointer

```
int const *p;
```

- The value p is pointing to can be changed
- The address p is pointing to cannot be changed

Pointer to constant

```
const int *p;
```

- The pointer p can point to anything
- What p points to cannot be changed

```
int a=0, b=1; const int *p1; int * const p2=&a;
p1=&a; cout << *p1 << *p2 << endl;
p1=&b; *p2=b; //p2=&b; *p1=b;
cout << *p1 << *p2 << endl;
```

References

Basics on references:

- Alias for another variable
- Changes on a reference are applied to the original variable
- Similar to a pointer that is automatically dereferenced
- Syntax: `int &a=3`

Remarks:

- Reference variable must be initialised
- The variable it refers to cannot be changed

References

Example.

ref.cpp

```cpp
#include <iostream>
using namespace std;
int square0(int x) {return x*x;}
void square1(int x, int &res) { res=x*x; }
//int& square2a(int x) { int b=x*x; return b; }
int& square2b(int x) { int b=x*x; int &res=b; return res; }
int& square2c(int x) { static int b=x*x; return b; }
int main () {
  int a=2;
  cout << square0(a) << ' ' << a << endl;
  square1(a,a); cout << a << endl;
  cout << square2b(a) << endl;
  cout << square2c(a) << endl;
}
```

The this pointer

The this keyword:

- Address of the object on which the member function is called
- Mainly used for disambiguation

boat.cpp

```cpp
#include <iostream>
using namespace std;
class Boat {
  public:
    Boat(string name, int tonnage, bool IsDocked) {
      this->name=name; this->tonnage=tonnage; this->IsDocked=IsDocked;
    }
    void dock() { IsDocked=1;  cout<<"Docked!\n"; }
    void undock() { IsDocked=0; cout<<"Undocked!\n"; }
  private: bool IsDocked; string name; int tonnage;
};
int main () {
  Boat b("abc",1234,1); b.undock();
}
```

Pointer to function

Similar to pointer to variables:

- Variable storing the address of a function
- Useful to give a function as argument to another function
- Useful for callback functions (e.g. GUI)

fctptr.c

```c
#include <stdio.h>
#include <string.h>
int gm(char *n) {
  printf("good morning %s\n",n);
  return strlen(n);
}
int main () {
  int (*gm_ptr)(char *)=gm;
  printf("%d\n",(*gm_ptr)("john"));
}
```

The enum and union keywords

enum_union.c

```c
#include<stdio.h>
typedef struct _activity {
  enum { BOOK, MOVIE, SPORT } type;
  union {
    int pages;
    double length;
    int freq;
  } prop;
} activity;
int main() {
  activity a[5];
  a[0].type=BOOK; a[0].prop.pages=192;
  a[1].type=SPORT; a[1].prop.freq=4;
  a[2].type=MOVIE; a[2].prop.pages=123;
  a[2].prop.length=92.5;
  printf("%f",a[2].prop.length);
}
```

The argc and *argv[] parameters

arg.c

```c
#include <stdio.h>
int main (int argc, char *argv[]) {
  printf ("program: %s\n",argv[0]);

  if (argc > 1) {
    for (int i=1; i<argc; i++)
      printf("argv[%d] = %s\n", i, argv[i]);
  }
  else printf("no argument provided\n");
  return 0;
}
```

Compilation process

Compilation is performed in three steps:

1. Pre-processing

```
sh $ gcc -E file.c
```

2. Assembling

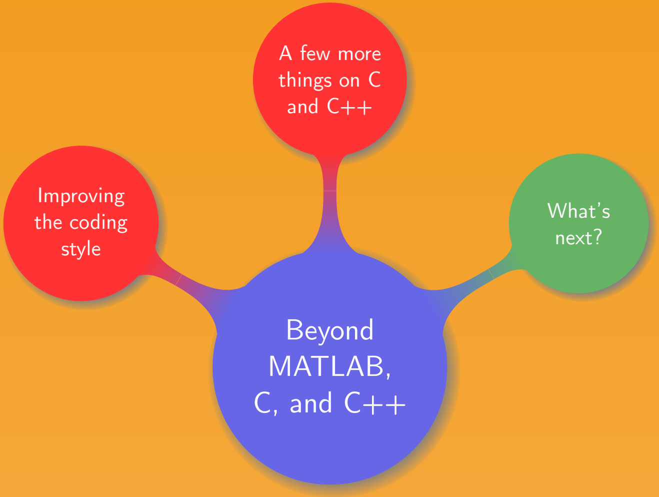
```
sh $ gcc -c file.c
```

3. Linking

```
sh $ gcc file.c
```

Commands at stage i performs stage 1 to i

Chapter organisation



Beyond MATLAB, C, and C++

Improving the coding style

A few more things on C and C++

What's next?

Present

- MATLAB:
  - Testing new algorithms
  - Getting quick results
- C:
  - Lower level
  - More complex, flexible
  - Faster, less base functions
- C++:
  - New programming strategy
  - Higher level
  - Convenient for big projects

Future

Important points that remain to be considered:

- More to learn on programming
- Languages of interest: C, Java, SQL, C++, PHP, CSS
- Other useful languages: Python, Perl, Ruby
- Designing a software: who is going to use it, where, how?
- More details on how computers are working
    - Data structures
    - Optimizations
    - How to improve efficiency

Key points

- Many things are left to learn
- Before coding always write an algorithm
- There no better way to learn than coding
- Do not reinvent the wheel, use libraries
- Each language has its own strengths, use them
- Extend your knowledge by building on what you already know

Thank you!