C++ Programming

Week 6: C++ Classes – Part I

Dr. Owen Chen
Cary Chinese School
Director of Math and Computer Science

2023 Summer

Week 6: Agenda

- Review Week 5 - Functions
- Review Homework 5)
- New Topic: C++ Classes

Function Review

A **function** (**procedure** or **routine**) is a piece of code that performs a *specific task.* Function is a block of code which only runs when it is called.

Purpose:

- **Avoiding code duplication**: less code for the same functionality ⟶ less bugs
- **Readability**: better express what the code does
- **Organization**: break the code in separate modules

Function Review

- Function is a block of code with a name.
- Declare a function with its name, parameters and return type
- Define a function with details
- Execute a function by calling the function
- A function takes zero or more arguments and usually returns a result.
- Functions can be overloaded, meaning that the same name may have different arguments and different return values

Function Declaration/Definition

Declaration/Prototype

A **declaration** (or *prototype*) of an entity is an identifier describing its type

A declaration is what the compiler and the linker needs to accept references (usage) to that identifier

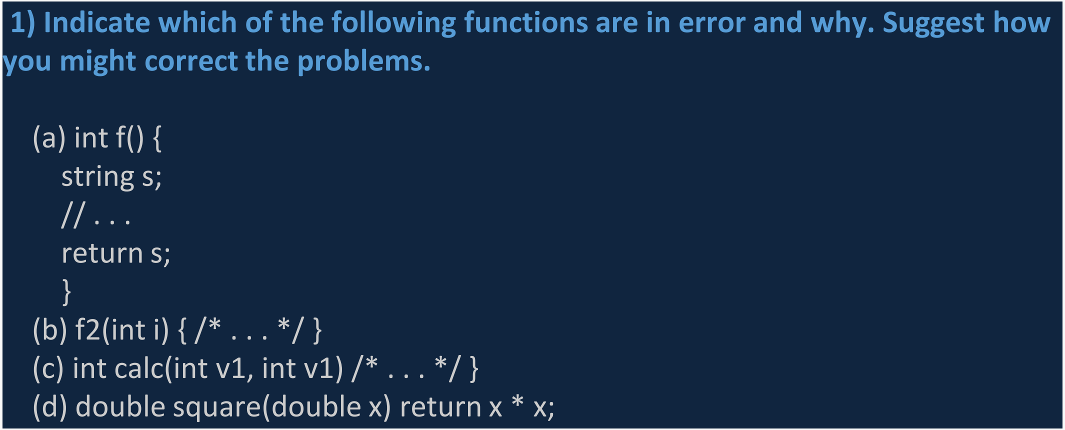
C++ entities (class, functions, etc.) can be declared multiple times (with the same signature)

Definition/Implementation

An entity **definition** is the implementation of a declaration

For each entity, only a single *definition* is allowed

Homework 5



1) Indicate which of the following functions are in error and why. Suggest how you might correct the problems.

```
(a) int f() {
       string s;
       // . . .
       return s;
       }
(b) f2(int i) { /* . . . */ }
(c) int calc(int v1, int v1) /* . . . */ }
(d) double square(double x) return x * x;
```

Homework 5

1) Indicate which of the following functions are in error and why. Suggest how you might correct the problems.

```
(a) int f() {.
    string s;
    // . . .
    return s;
    }

(b) f2(int i) { /* . . . */ }
(c) int calc(int v1, int v1) /* . . . */ }
(d) double square(double x) return x * x;
```

Solutions to 1):

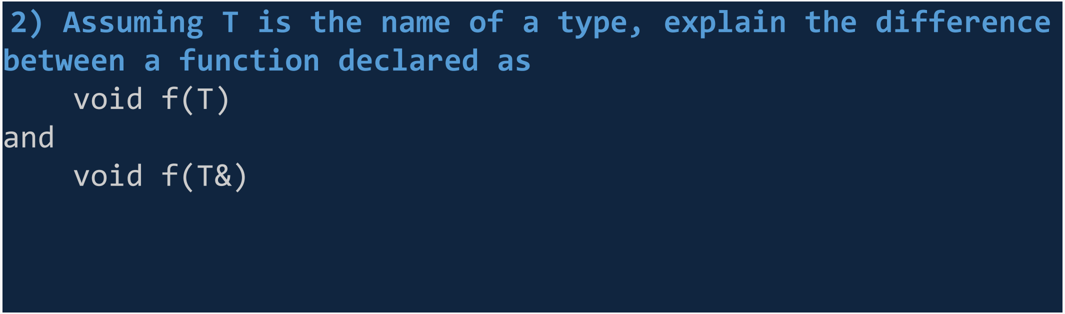
(a) Return type and the actual return value mismatch.

(b) No return type. Add a void keyword.

(c) Two input arguments are both named v1

(d) Missing {}

Homework 5



```
 2) Assuming T is the name of a type, explain the difference
between a function declared as
    void f(T)
and
    void f(T&)
```

Homework 5

2) Assuming T is the name of a type, explain the difference between a function declared as

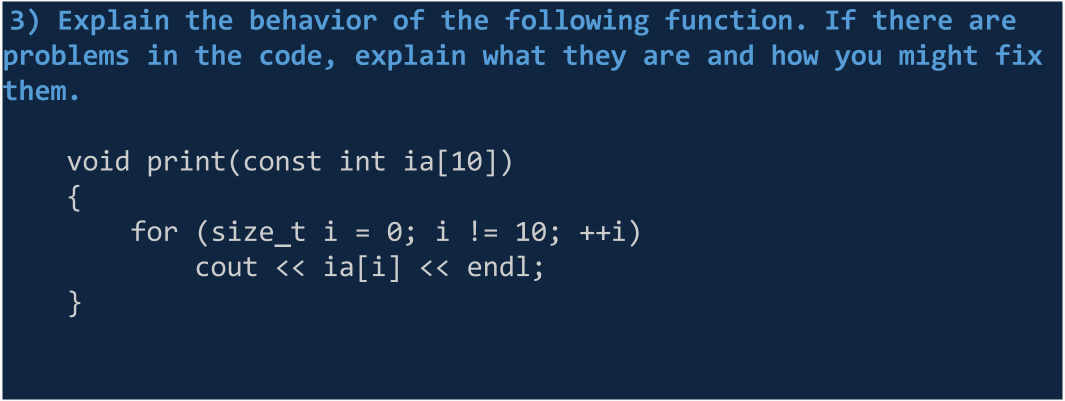
```
    void f(T)
and
    void f(T&)
```

Solutions to 2):

f(T) is a function that passes its argument by value.

f(T&) is a function that passes its argument by reference.

Homework 5



3) Explain the behavior of the following function. If there are problems in the code, explain what they are and how you might fix them.

```
void print(const int ia[10])
{
    for (size_t i = 0; i != 10; ++i)
        cout << ia[i] << endl;
}
```

Homework 5

Solutions to 3):

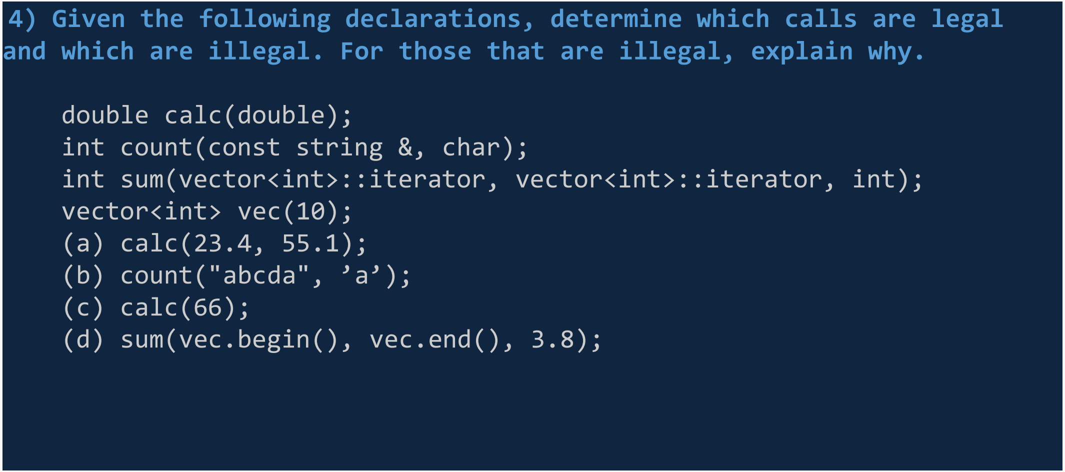
The function **print** takes as an argument an array of 10 constant integers. The function then iterates through the array using a for loop and prints each element of the array to the standard output stream, followed by a newline character.

However, there is a problem with the function's parameter declaration. The parameter const int ia[10] is misleading because it suggests that the function will only accept arrays of size 10. In reality, the parameter is equivalent to const int* ia, which means that the function will accept a pointer to a constant integer. This can lead to problems if an array of a different size is passed to the function, as the function will still try to access 10 elements, potentially causing undefined behavior.

One way to fix this issue is to change ia to a pointer and add a size of the ia array:

```cpp
void print(const int ia[], int size)
{
    for (size_t i = 0; i < size; ++i)
        cout << ia[i] << endl;
}
```

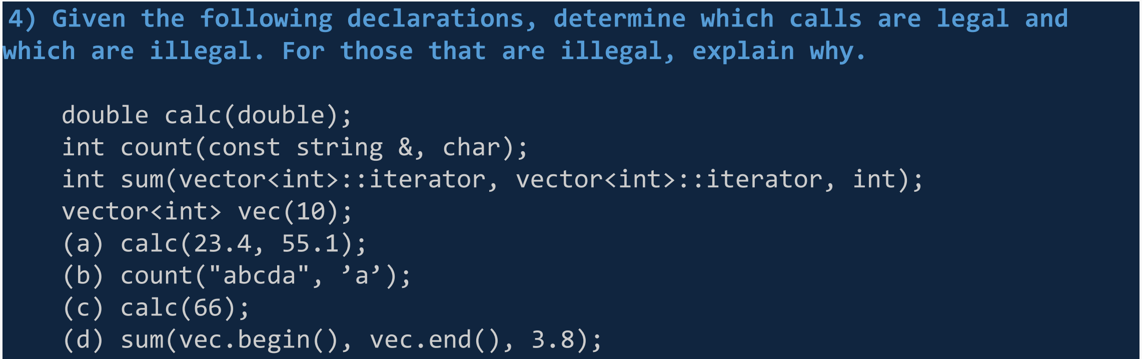
Homework 5



4) Given the following declarations, determine which calls are legal and which are illegal. For those that are illegal, explain why.

```
    double calc(double);
    int count(const string &, char);
    int sum(vector<int>::iterator, vector<int>::iterator, int);
    vector<int> vec(10);
    (a) calc(23.4, 55.1);
    (b) count("abcda", ’a’);
    (c) calc(66);
    (d) sum(vec.begin(), vec.end(), 3.8);
```

Homework 4



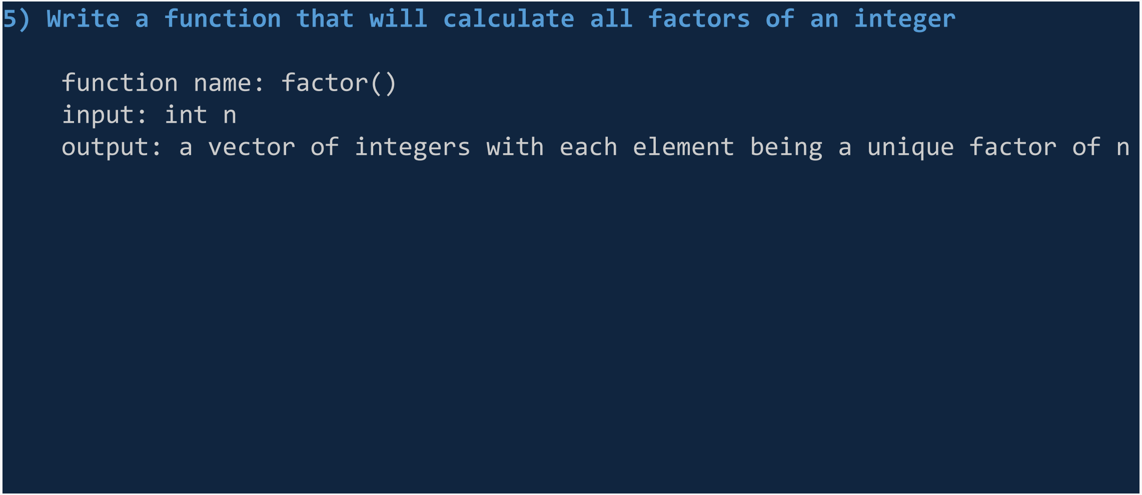
4) Given the following declarations, determine which calls are legal and which are illegal. For those that are illegal, explain why.

```
    double calc(double);
    int count(const string &, char);
    int sum(vector<int>::iterator, vector<int>::iterator, int);
    vector<int> vec(10);
    (a) calc(23.4, 55.1);
    (b) count("abcda", ’a’);
    (c) calc(66);
    (d) sum(vec.begin(), vec.end(), 3.8);
```

Solutions to 4)

a) **Illegal. Pass two values to a function has a single parameter.**
b) **Legal. Pass a string literal to a string reference is fine.**
c) **Legal. Convert 66 to a double as a parameter value in calc()**
d) **Legal. 3.8 is converted to integer. However, the outcome is random as vec is not initialized.**

Homework 5



```
5) Write a function that will calculate all factors of an integer

    function name: factor()
    input: int n
    output: a vector of integers with each element being a unique factor of n
```

Homework 5

```
6) Write a function that will calculate the GCD (greatest common divisor) of two integers.
Use factor() function defined in 5)

   function name: gcd()
   input: int a, int b
   output: gcd of (a, b)

7) Write a function that determines whether an input integer is a prime or not.

   function name: isPrime()
   input: int
   output: bool

8) Write a program that will produce a list of prime numbers that are less than a given
input integer. Use isPrime() function in 7) if needed.

   function name: prime_list()
   input: int n
   output: a list of prime numbers

 Main program: prompt user to enter a number and store it as integer n.
 Call prime_list() and print the list of primes on screen.
```

C++ Classes

C++ Classes and Objects

- C++ is an object-oriented programming language.
- Everything in C++ is associated with classes and objects, along with its attributes and methods.
- For example: a car is an **object**.
  - **- attributes**, such as weight and color
  - **- methods**, such as drive and brake.
- Attributes and methods are basically **variables** and **functions** that belongs to the class. These are often referred to as "**class members**".

C/C++ Structure

Before C++, C has a **structure** (`struct`) is a collection of variables of the same or different data types under a single name

Structures

A struct in **C** is a type consisting of a sequence of data members.

Some functions/statements are needed to operate the data members of an object of a struct type.

```
struct Student
{
   char name[20];
   int born;
   bool male;
};

struct Student stu;
strcpy(stu.name, “John");
stu.born = 2000;
stu.male = true;
```

C++ Classes

- In C++, we define our own data structures by defining a **class**.
- A class defines a type along with a collection of operations that are related to that type.
- The class mechanism is one of the most important features in C++.

C++ Classes

- Classes are an expanded concept of data structures: like data structures, they can contain data members, but they can also contain functions as members.

- An object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

- Classes are defined using either keyword class or keyword struct.

C++ Classes

To use a class, we need to know three things:

- What is its name?
- Where is it defined?
- What operations does it support?

Struct vs Class

C/C++ Structure

A **structure** (`struct`) is a collection of variables of the same or different data types under a single name

C++ Class

A **class** (`class`) extends the concept of structure to hold functions as members

`struct` vs. `class`

Structures and classes are *semantically* equivalent. In general, `struct` represents *passive* objects, while `class` *active* objects

C++ Class Members - Data and Function Members

Data Member

Data within a class are called **data members** or **class fields**

Function Member

Functions within a class are called **function members** or **methods**

struct Declaration and Definition

struct declaration

```
struct A;      // struct declaration
```

struct definition

```
struct A {     // struct definition
   int x;      // data member
   string name;
   void f(); // function member
};
```

class Declaration and Definition

class declaration

```
class A;        // class declaration
```

class definition

```
class A {       // class definition
   int x;       // data member
   string name;
  void f(); // function member
};
```

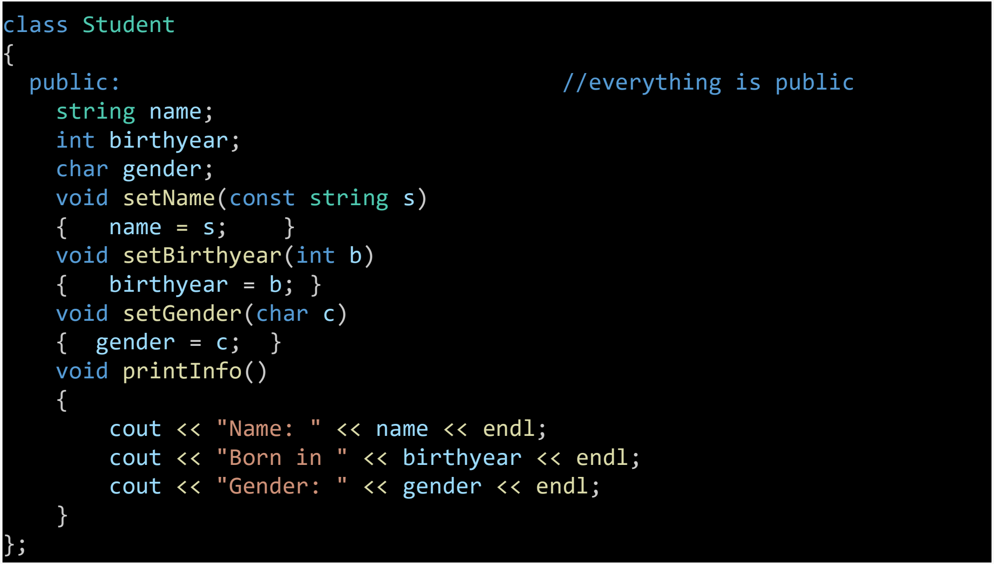
Class Function Declaration and Definition

```
class A {
   void g();          // function member declaration

   void f() {         // function member declaration
       cout << "f"; // inline definition
   }
};

void A::g() {         // function member definition
    cout << "g";      // out side definition
}
```

class Members

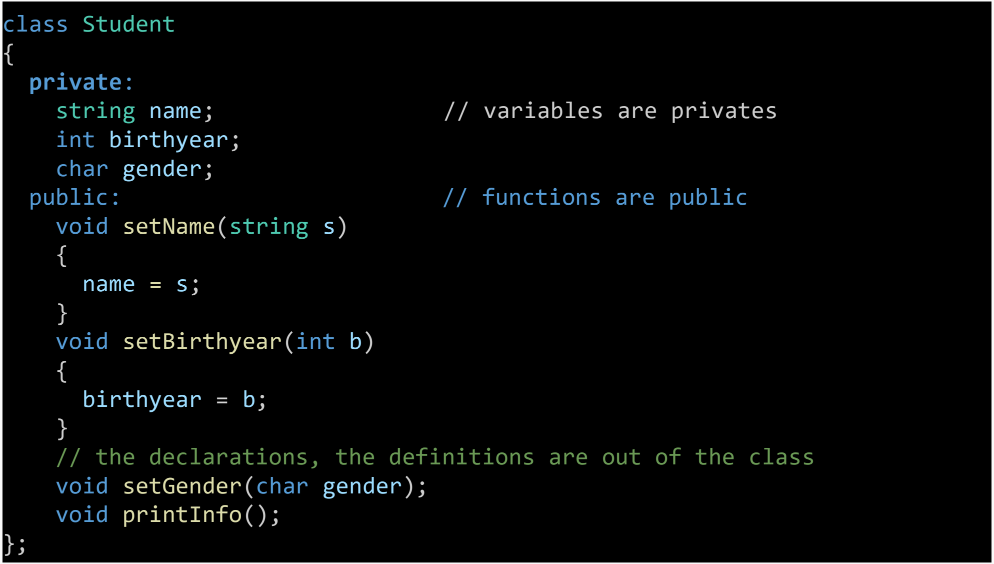
```
class B {
    void g() { cout << "g"; } // function member
};

 class A {
    int  x;                // data member
    B    b;                // data member b is a class of B
    void f() { cout << "f"; } // function member
};

A a;
a.x;
a.f();
a.b.g();
```

C++ class Example: firstclass.cpp



```cpp
class Student
{
  public:                                 //everything is public
    string name;
    int birthyear;
    char gender;
    void setName(const string s)
    {   name = s;    }
    void setBirthyear(int b)
    {   birthyear = b; }
    void setGender(char c)
    {  gender = c;  }
    void printInfo()
    {
        cout << "Name: " << name << endl;
        cout << "Born in " << birthyear << endl;
        cout << "Gender: " << gender << endl;
    }
};
```

C++ class Example: student.h



```cpp
class Student
{
  private:
    string name;                 // variables are privates
    int birthyear;
    char gender;
  public:                        // functions are public
    void setName(string s)
    {
      name = s;
    }
    void setBirthyear(int b)
    {
      birthyear = b;
    }
    // the declarations, the definitions are out of the class
    void setGender(char gender);
    void printInfo();
};
```

C++ class Example: student.cpp

```
//Class Definitions

#include <iostream>
#include <cctype>
#include "student.h"

void Student::setGender(char g)
{
   gender = tolower(g);
}

void Student::printInfo()
{
   cout << "Name: " << name << endl;
   cout << "Born in year " << born << endl;
   cout << "Gender: " << (gender==‘m’? "Male":gender==‘f’? "Female“:“Other”) << endl;
}
```

C++ class Example: student_main.cpp

```
//Main Program – instantiate a Class and call member functions

#include "student.h"

int main(){
   Student st1;
   st1.setName(“John");
   st1.setBirthyear(2008);
   st1.setGender(‘m’);
   st1.printInfo();
   return 0;
}
```

Source Code Management

The source code can be saved into multiple files. Create a makefile to link them.

student.h

```
class Student
{
 private:
   string name;
   int birthyear;
   char gender;
 public:
   void setName(string s) // inline definition
   {
      name = s;
   }
   void setBirthyear(int y) // inline definition
   {
      birthyear = y;
   }
   void setGender(char g);
   void printInfo();
};
```

student.cpp

```
#include <iostream>
#include <cctype>
#include "student.h"

void Student::setGender(char g)
{
   gender = tolower(g);
}

void Student::printInfo()
{
   cout << "Name: " << name << endl;
   cout << "Born in year " << born << endl;
   cout << "Gender: " << (gender==‘m’?
"Male":gender==‘f’? "Female“:“Other”) << endl;
}
```

Student_main.cpp

```
#include "student.h"
int main(){
   Student st1;
   ...
   st1.printInfo();
   return 0;
}
```

Makefile

Compile multiple dependent source code files

- When there are multiple C++ source code files, compile each cpp file into an object first with this syntax:
  - **g++ -c program1.cpp   -o program1.o**
  - **g++ -c program2.cpp   -o program2.o**

- Then link objects together:
  - **g++ program1.o program2.o -o program.exe**

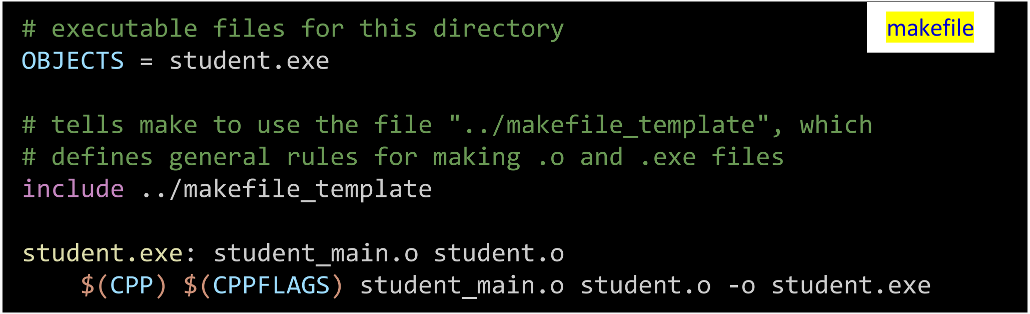
Compile multiple dependent source code files

- In our student example:

```
$ g++ -c student.cpp –o student.o

$ g++ -c student_main.cpp –o student_main.o

$ g++ student.o student_main.o –o student.exe
```

Makefile is another method to compile multiple files

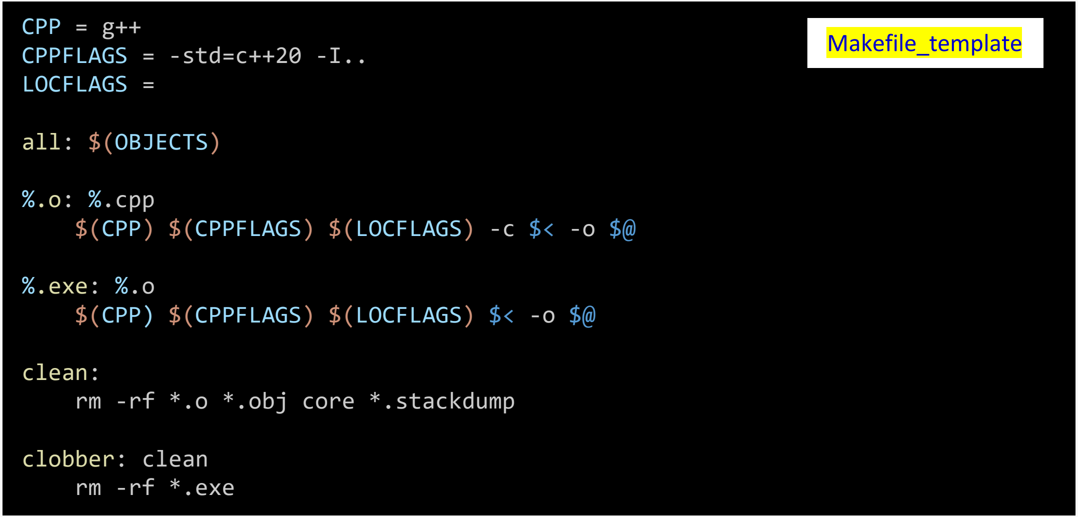
Create a makefile to compile multiple files



makefile

```
# executable files for this directory
OBJECTS = student.exe

# tells make to use the file "../makefile_template", which
# defines general rules for making .o and .exe files
include ../makefile_template

student.exe: student_main.o student.o
    $(CPP) $(CPPFLAGS) student_main.o student.o -o student.exe
```

Make file template for g++



```
CPP = g++
CPPFLAGS = -std=c++20 -I..
LOCFLAGS =

all: $(OBJECTS)

%.o: %.cpp
    $(CPP) $(CPPFLAGS) $(LOCFLAGS) -c $< -o $@

%.exe: %.o
    $(CPP) $(CPPFLAGS) $(LOCFLAGS) $< -o $@

clean:
    rm -rf *.o *.obj core *.stackdump

clobber: clean
    rm -rf *.exe
```

Makefile_template

Make file commands

The following commands can be used with this makefile:

```
$ make
$ make all
$ make clean
$ make clobber

$ make student.exe
```