

# ECOTE - preliminary project

**Semester:** 18L

**Author:** Assam Chaudhary

**Subject:** Write a program reading C++ code and constructing class inheritance tree

## General overview and assumptions

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

When deriving a class from a base class, the base class may be inherited through **public**, **protected** or **private** inheritance. A C++ class can also inherit members from more than one class, this is called **multiple inheritance**.

The program will accept text files containing **C++** source code. Goal of the project is to **identify all class declarations and construct an inheritance tree for classes**. I assume that the input code may **not** be syntactically correct in terms of class declarations.

**Class declarations in C++ could be in the following ways:**

Simple class declaration

```
class Base {  
....  
};
```

Inherited class

```
class Derived: Base{  
...  
};
```

Multiple inheritance

```
class Derived_2: public Base, public Derived {  
...  
};
```

Nested classes and different scopes

```
class A{  
...  
    class B{        //local class B  
        ...  
    };  
};
```

```
class B {      //global class B
...
};
```

Forward declaration

```
class A;
...
class A{
...
};
```

## Functional requirements

The program should perform a lexical analysis on the input **C++** source code, splitting the code line by line into **tokens** and classifying keywords and symbols. After analysing that the code is syntactically correct, amongst these classified tokens, the program should be able to identify class declarations and keep track of base and derived classes.

The program should also ignore **class** declarations in **comments** (both line and block) as well as any mention of the keyword **class** in a **string**.

The output would be names of each class with the line number declared on and its parent classes also with line number declared on.

The program should also take care of class scope, where more than one classes with the same name can exist in different scopes (local or global).

## Detailed syntax of the input language

class-name  
    identifier

enum-name  
    identifier

typedef-name  
    identifier

### Expressions

expression  
    assignment-expression  
    expression , assignment-expression

assignment-expression  
    conditional-expression  
    unary-expression assignment-operator assignment-expression

assignment-operator  
    =  
    \* =

`/=`  
`%=`  
`+=`  
`-=`  
`<<=`  
`>>=`  
`&=`  
`^=`  
`|=`

`conditional-expression`  
    `logical-OR-expression`  
    `logical-OR-expression` ? `expression` : `conditional-expression`

`logical-OR-expression`  
    `logical-AND-expression`  
    `logical-OR-expression` || `logical-AND-expression`

`logical-AND-expression`  
    `inclusive-OR-expression`  
    `logical-AND-expression` && `inclusive-OR-expression`

`inclusive-OR-expression`  
    `exclusive-OR-expression`  
    `inclusive-OR-expression` | `exclusive-OR-expression`

`exclusive-OR-expression`  
    `AND-expression`  
    `exclusive-OR-expression` ^ `AND-expression`

`AND-expression`  
    `equality-expression`  
    `AND-expression` & `equality-expression`

`equality-expression`  
    `relational-expression`  
    `equality-expression` == `relational-expression`  
    `equality-expression` != `relational-expression`

`relational-expression`  
    `shift-expression`  
    `relational-expression` < `shift-expression`  
    `relational-expression` > `shift-expression`  
    `relational-expression` <= `shift-expression`  
    `relational-expression` >= `shift-expression`

`shift-expression`  
    `additive-expression`  
    `shift-expression` << `additive-expression`  
    `shift-expression` >> `additive-expression`

`additive-expression`  
    `multiplicative-expression`

additive-expression + multiplicative-expression  
additive-expression - multiplicative-expression

multiplicative-expression  
pm-expression  
multiplicative-expression \* pm-expression  
multiplicative-expression / pm-expression  
multiplicative-expression % pm-expression

pm-expression  
cast-expression  
pm-expression .\* cast-expression  
pm-expression ->\* cast-expression

cast-expression  
unary-expression  
( type-name ) cast-expression

unary-expression  
posfix-expression  
++ unary-expression  
-- unary-expression  
unary-operator cast-expression  
sizeof unary-expression  
sizeof ( type-name )  
allocation-expression  
deallocation-expression

unary-operator  
\*  
&  
+  
-  
!  
~

allocation-expression  
:: new placement new-type-name new-initializer  
new placement new-type-name new-initializer  
:: new new-type-name new-initializer  
new new-type-name new-initializer  
:: new placement new-type-name  
new placement new-type-name  
:: new new-type-name  
new new-type-name  
:: new placement ( type-name ) new-initializer  
new placement ( type-name ) new-initializer  
:: new ( type-name ) new-initializer  
new ( type-name ) new-initializer  
:: new placement ( type-name )  
new placement ( type-name )  
:: new ( type-name )  
new ( type-name )

```

placement
    ( expression-list )

new-type-name
    type-specifier-list new-declarator
    type-specifier-list

new-declarator
    * cv-qualifier-list new-declarator
    * new-declarator
    * cv-qualifier-list
    *
    complete-class-name :: * cv-qualifier-list new-declarator
    complete-class-name :: * new-declarator
    complete-class-name :: * cv-qualifier-list
    complete-class-name :: *
    new-declarator [ expression ]
    [ expression ]

new-initializer
    ( initializer-list )
    ( )

deallocation-expression
    :: delete cast-expression
    delete cast-expression
    :: delete [ ] cast-expression
    delete [ ] cast-expression

postfix-expression
    primary-expression
    postfix-expression [ expression ]
    postfix-expression ( expression-list )
    postfix-expression ( )
    simple-type-name ( expression-list )
    simple-type-name ( )
    postfix-expression . name
    postfix-expression -> name
    postfix-expression ++
    postfix-expression --

expression-list
    assignment-expression
    expression-list , assignment-expression

primary-expression
    literal
    this
    :: identifier
    :: operator-function-name
    :: qualified-name
    ( expression )

```

name

name

- identifier
- operator-function-name
- conversion-function-name
- ~ class-name
- qualified-name

qualified-name

- qualified-class-name :: name

literal

- integer-constant
- character-constant
- floating-constant
- string-literal

## **Declarations**

declaration

- decl-specifiers declarator-list ;
- decl-specifiers ;
- declarator-list ;
- function-definition
- template-declaration
- linkage-specification

decl-specifier

- storage-class-specifier
- type-specifier
- fct-specifier
- friend
- typedef

decl-specifiers

- decl-specifiers decl-specifier
- decl-specifier

storage-class-specifier

- auto
- register
- static
- extern

fct-specifier

- inline
- virtual

type-specifier

- simple-type-name
- class-specifier
- enum-specifier

elaborated-type-specifier  
const  
volatile

simple-type-name  
complete-class-name  
qualified-type-name  
char  
short  
int  
long  
signed  
unsigned  
float  
double  
void

elaborated-type-specifier  
class-key identifier  
class-key class-name  
enum enum-name

class-key  
class  
struct  
union

qualified-type-name  
typedef-name  
class-name :: qualified-type-name

complete-class-name  
qualified-class-name  
:: qualified-class-name

qualified-class-name  
class-name  
class-name :: qualified-class-name

enum-specifier  
enum identifier { enum-list }  
enum { enum-list }  
enum identifier { }  
enum { }

enum-list  
enumerator  
enum-list , enumerator

enumerator  
identifier  
identifier = constant-expression

constant-expression  
conditional-expression

linkage-specification  
extern string-literal { declaration-list }  
extern string-literal { }  
extern string-literal declaration

declaration-list  
declaration  
declaration-list declaration

## **Declarators**

declarator-list  
init-declarator  
declarator-list , init-declarator

init-declarator  
declarator initializer  
declarator

declarator  
dname  
ptr-operator declarator  
declarator ( argument-declaration-list ) cv-qualifier-list  
declarator ( argument-declaration-list )  
declarator [ constant-expressio ]  
declarator [ ]  
( declarator )

ptr-operator  
\* cv-qualifier-list  
\*  
& cv-qualifier-list  
&  
complete-class-name :: \* cv-qualifier-list  
complete-class-name :: \*

cv-qualifier-list  
cv-qualifier cv-qualifier-list  
cv-qualifier

cv-qualifier  
const  
volatile

dname  
name  
class-name  
~ class-name  
typedef-name



qualified-type-name

type-name

type-specifier-list abstract-declarator

type-specifier-list

type-specifier-list

type-specifier type-specifier-list

type-specifier

abstract-declarator

ptr-operator abstract-declarator

ptr-operator

abstract-declarator ( argument-declaration-list ) cv-qualifier-list

( argument-declaration-list ) cv-qualifier-list

abstract-declarator ( argument-declaration-list )

( argument-declaration-list )

abstract-declarator [ constant-expression ]

[ constant-expression ]

abstract-declarator [ ]

[ ]

( abstract-declarator )

argument-declaration-list

arg-declaration-list ...

...

arg-declaration-list , ...

arg-declaration-list

argument-declaration

arg-declaration-list , argument-declaration

argument-declaration

decl-specifiers declarator

decl-specifiers declarator = expression

decl-specifiers abstract-declarator

decl-specifiers

decl-specifiers abstract-declarator = expression

decl-specifiers = expression

function-definition

decl-specifiers declarator ctor-initializer fct-body

declarator ctor-initializer fct-body

decl-specifiers declarator fct-body

declarator fct-body

fct-body

compound-statement

initializer

= assignment-expression

= { initializer-list }

```
= { initializer-list , }  
( expression-list )
```

```
initializer-list  
    assignment-expression  
    initializer-list , assignment-expression  
    { initializer-list }  
    { initializer-list , }
```

#### # r.17.5 Class Declarations

```
class-specifier  
    class-head { member-list }  
    class-head { }
```

```
class-head  
    class-key identifier base-spec  
    class-key base-spec  
    class-key identifier  
    class-key  
    class-key class-name base-spec  
    class-key class-name
```

```
member-list  
    member-declaration member-list  
    member-declaration  
    access-specifier : member-list  
    access-specifier :
```

```
member-declaration  
    decl-specifiers member-declarator-list ;  
    member-declarator-list ;  
    decl-specifiers ;  
    ;  
    function-definition ;  
    function-definition  
    qualified-name ;
```

```
member-declarator-list  
    member-declarator  
    member-declarator-list , member-declarator
```

```
member-declarator  
    declarator pure-specifier  
    declarator  
    identifier : constant-expression  
    : constant-expression
```

```
pure-specifier  
    = integer-constant
```

```
base-spec  
    : base-list
```

base-list  
    base-specifier  
    base-list , base-specifier

base-specifier  
    complete-class-name  
    virtual access-specifier complete-class-name  
    virtual complete-class-name  
    access-specifier virtual complete-class-name  
    access-specifier complete-class-name

access-specifier  
    private  
    protected  
    public

conversion-function-name  
    operator conversion-type-name

conversion-type-name  
    type-specifier-list ptr-operator  
    type-specifier-list

ctor-initializer  
    : mem-initializer-list

mem-initializer-list  
    mem-initializer  
    mem-initializer , mem-initializer-list

mem-initializer  
    complete-class-name ( expression-list )  
    complete-class-name ( )  
    identifier ( expression-list )  
    identifier ( )

operator-function-name  
    operator operator-name

operator-name  
    new  
    delete  
    +  
    -  
    \*  
    /  
    %  
    ^  
    &  
    |  
    ~  
    !

=  
<  
>  
+=  
-=  
\*=  
/=  
%=  
^=  
&=  
~=  
<<  
>>  
>>=  
<<=  
==  
!=  
<=  
>=  
&&  
||  
++  
--  
,  
->\*  
->  
( )  
[ ]

## Statements

statement

labeled-statement  
expression-statement  
compound-statement  
selection-statement  
iteration-statement  
jump-statement  
declaration-statement  
try-block

labeled-statement

identifier : statement  
case constant-expression : statement  
default : statement

expression-statement

expression ;  
;

compound-statement

{ statement-list }  
{ }

```

statement-list
    statement
    statement-list statement

selection-statement
    if ( expression ) statement
    if ( expression ) statement else statement
    switch ( expression ) statement

iteration-statement
    while ( expression ) statement
    do statement while ( expression ) ;
    for ( for-init-statement expression ; expression ) statement
    for ( for-init-statement ; expression ) statement
    for ( for-init-statement expression ; ) statement
    for ( for-init-statement ; ) statement

for-init-statement
    expression-statement
    declaration-statement

jump-statement
    break ;
    continue ;
    return expression ;
    return ;
    goto identifier ;

declaration-statement
    declaration

```

## **Templates**

```

template-declaration
    template < template-argument-list > declaration

template-argument-list
    template-argument
    template-argument-list , template-argument

template-argument
    type-argument
    argument-declaration

type-argument
    class identifier

template-class-name
    template-name < template-arg-list >

template-arg-list
    template-arg

```

```
template-arg-list , template-arg
```

```
template-arg  
    expression  
    type-name
```

## Exception Handling

```
try-block  
    try compound-statement handler-list
```

```
handler-list  
    handler handler-list  
    handler
```

```
handler  
    catch ( exception-declaration ) compound-statement
```

```
exception-declaration  
    type-specifier-list declarator  
    type-specifier-list abstract-declarator  
    type-specifier-list  
    ...
```

```
throw-expression  
    throw expression  
    throw
```

```
exception-specification  
    throw ( type-list )  
    throw ( )
```

```
type-list  
    type-name  
    type-list , type-name
```

## Implementation

### General architecture

The project is implemented in **python 3**. The libraries used are **re** (Regular expressions). The program reads c++ source files, finds classes declared and outputs them.

### Data structures

The data structures used are **array** (python list and tuples), **hash table** (python dictionary) and a custom **classInfo** class and a **set** of **classInfo** objects.

```
class classInfo:  
    def __init__(self, name, line_no, scope):  
        self.name = name
```

```
self.line_no = line_no
self.parents = set()
self.objects = set()
self.children = set()
self.scope = scope
```

## Module descriptions

### Module 1

The **first** part of the program reads the file and splits it line by line. Each individual line is then further split into **tokens** which are stored in form of a list/array. **Hash tables** with different operators, keywords etc are defined with their **keys** being the keywords or operators and **values** being their description. It acts as a **symbol table**.

The **tokens** are input into a **syntax analyzer** which checks if the code is syntactically correct. If there is any syntax error, the code is **rejected** and the program terminates with an error message. If the code passes the **syntax test**, the program moves to the next module.

### Module 2

The **second module** deals with the comments. This could be both single line or block. If the program encounters a single line comment it ignores everything starting from the comment until the end of line . If a start of a block comment is encountered then it looks for the end of the block comment and ignores everything in between.

### Module 3

Now we have the actual source code (without any comments). The **third module** looks for the keyword **class**, all the class declarations in the code. It also deals with the situation where the word class is written in quotation marks ("....class....") in case of a value assigned to some string variable in which case the program ignores it.

Once the class keyword is encountered, it is added into a set of of **classInfo** objects along with it line number and the name of it's parent classes if any and the line numbers of the parent classes.

## Input/output description

The input is various parts of a C++ code. It contains several types class declarations (mentioned earlier) as well as comments containing "class" keywords. Some of the files do not contain this keyword as well as incorrect class declarations which would give an error as well as files containing same class names but in different scopes as well as in the same scope which should of course give an error.

The output is a set of classes with their names and line number on which it was declared and the name of its parents if any along with the line number on which the parents were declared. If there were no classes in the program, it tells the user that no classes were found. If there was a syntax issue with class declaration, then an error message is outputted with number of line with a syntax issue.

## Functional test cases

1. Input: `find_classes('sample.cpp')`  
Output: Running on file: `sample.cpp`

```
class "nested" declared on line 84
parent class(es):
Shape (Line 15)
```

```
class "Shape2" declared on line 60
parent class(es):
Shape (Line 15)
Rectangle (Line 41)
```

```
class "Rectangle" declared on line 41
parent class(es):
Shape (Line 15)
```

```
class "a" declared on line 92
parent class(es):
nested (Line 84)
```

```
class "nested" declared on line 69
parent class(es):
Shape2 (Line 60)
```

```
class "Shape" declared on line 15
```

```
class "base" declared on line 65
```

```
class "Rec" declared on line 74
parent class(es):
Shape (Line 15)
```

```
class "nested" declared on line 21
```

```
class "Rec" declared on line 54
parent class(es):
Rectangle (Line 41)
```

2. Input: `find_classes('string.cpp')`  
Output: Running on file: `string.cpp`



No classes found

3. Input: find\_classes('comments.cpp')  
Output: Running on file: comments.cpp  
No classes found

4. Input: find\_classes('nested.cpp')  
Output: Running on file: nested.cpp

class "Shape" declared on line 34

class "Rec" declared on line 7

class "b" declared on line 18

class "nested" declared on line 26  
parent class(es):  
base (Line 5)

class "nested" declared on line 43  
parent class(es):  
base (Line 5)

class "c" declared on line 22

class "base" declared on line 5

class "nested" declared on line 11

5. Input: find\_classes('wrong\_keyword.cpp')  
Output: Running on file: wrong\_keyword.cpp  
Line: 23 - Class "pub" does not exist

6. Input: find\_classes('nonexist\_class.cpp')  
Output: Running on file: nonexist\_class.cpp  
Line: 5 - Class "Shape" does not exist

7. Input: find\_classes('noClassName.cpp')  
Output: Running on file: noClassName.cpp  
Line: 6 - Incorrect class declaration

8. Input: find\_classes('noColon.cpp')  
Output: Running on file: noColon.cpp  
Line: 11 - Incorrect class declaration

9.    Input: `find_classes('imbalancedParan.cpp')`  
      Output: Running on file: `imbalancedParan.cpp`  
              Imbalanced parentheses in class declarations
  
10.   Input: `find_classes('frwd_dec.cpp')`  
      Output: Running on file: `frwd_dec.cpp`  
  
          class "Object" declared on line 7  
  
          class "World" declared on line 5