**BACHELOR OF ENGINEERING (HONOURS) IN ELECTRONIC ENGINEERING**

**ACADEMIC YEAR 2018/2019**

**Software development 2 Assignment on an application for Radio Station**

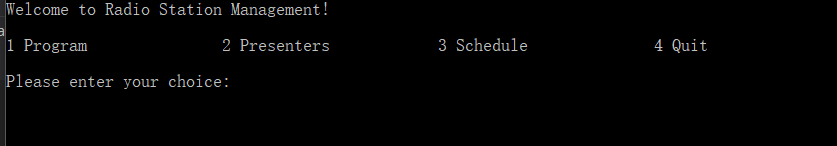


### Author: Jiong Du

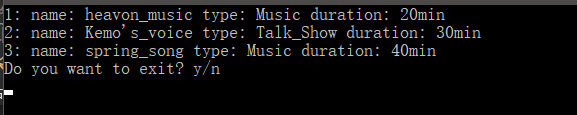
### Student number: X00142657

# Description:

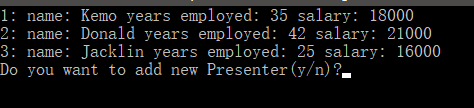
This project is about an application for radio station management. In the menu, you can choose 1, 2, 3 to see the information about programs, presenters and schedule. The information in program option is program title, program type and duration. The presenter option describes presenter name, years employed and salaries. The schedule option details program name, presenter name, audience number, advertising, start time of the program and last time of the ad. After choosing the schedule, you can enter a number to select the program you want to see and view the details of this program or the presenters’ details of this program.



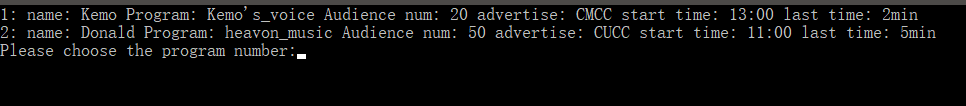
**Figure 1 main menu**



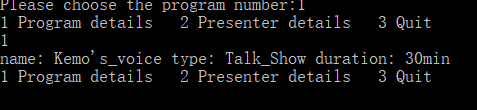
**Figure 2 program option**

****

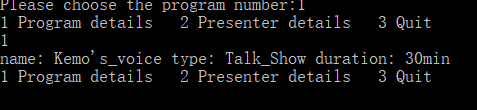
**Figure 3 presenter option**

****

**Figure 4 schedule option**

****

**Figure 5 program details of program 1**

****

**Figure 6 presenter details of program 1**

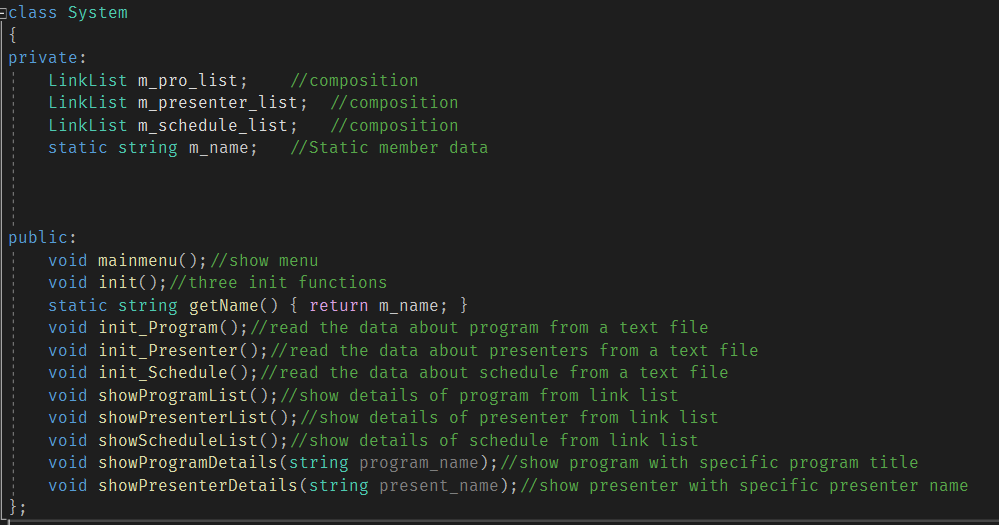
A Program class is chosen to own the basic information of a radio program, title and duration. A Music class and a TalkShow class inheriting from the Program class represent the specific radio programs, they both have the additional attribute, type, to note the type of the program. The project is required to use queue link list, so a LinkList class and a Node class are chosen to be created.

It should be noticed that because three types of data, which are program, presenter and schedule need to be added into the link list, the ‘data’ in the Node class is created as void\* so this pointer can point to any types of data.

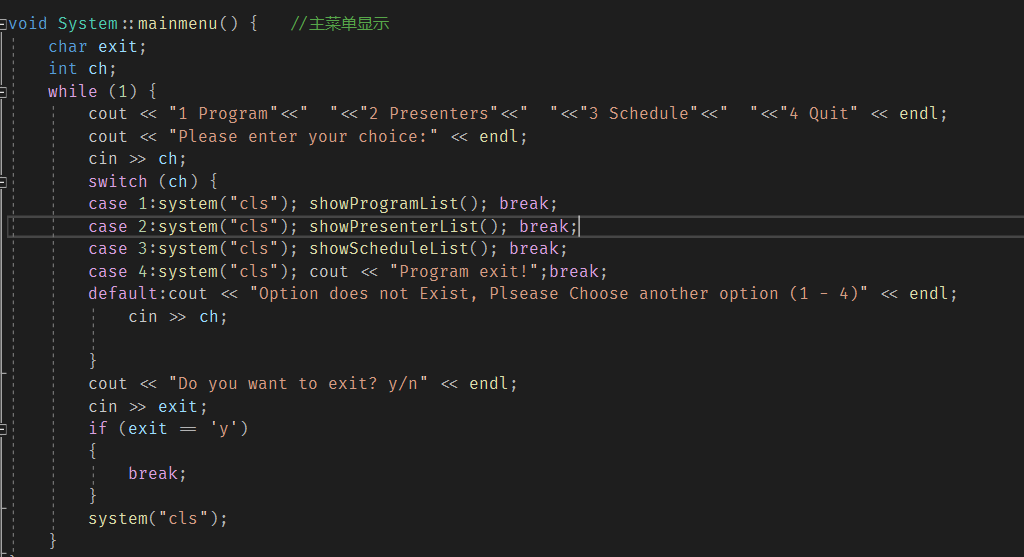
A Schedule class is used to show the timetable of programs including advertisements. A System class is selected as the core of the project, it is used to read data from text files into link lists and print the information from link lists.

Association and operator overloading as a member function are not clear in the described application, so additional methods are used in main to show them.

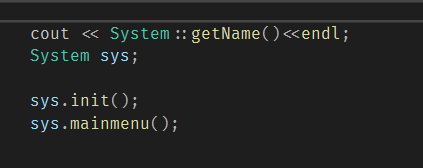
**[1]** A basic class, with access member functions, Constructors, Static member data.

**(a)** 

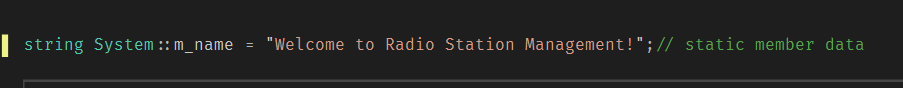
**Figure 1.1a System class**

****

**Figure 1.2a main menu function**

****

**Figure 1.3a**

****

**Figure 1.4a**

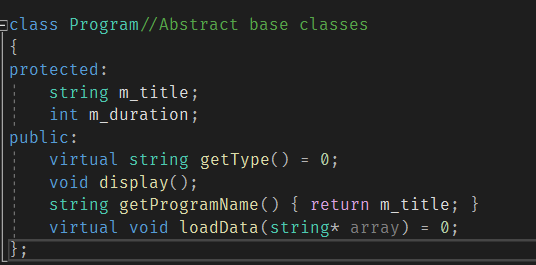
**(b)**

In Figure 1.1a, the System class is a basic class with access member function, constructors and static member data. This class uses the default constructor, mainmenu is a member function and it is used to show the main menu of radio station management. It is called by an object ‘sys’, which is created in main. The System class also has a static member data ‘m\_name’ and it is given the value in System.cpp. when there is ‘static’ before the data , it usually means that this is a static data.

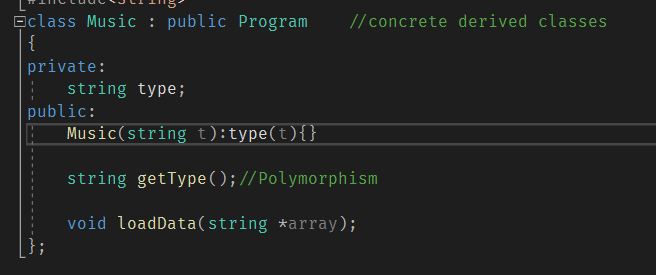
**(c)**

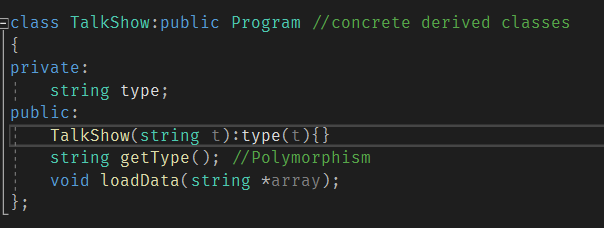
Access member functions are functions that can be called directly by objects of the class. In System class, all the functions are access member function because they can be called by the objects of this class. in Figure 1.3a, init and mainmenu functions are called by ‘sys’ in main. Constructors are called when the object of the class is created, here, the default constructor is used. The Static member data is owned to the class and all the objects share the static data. There is also a static member function getName and this function can be called by class name or objects, but the value is the same. In Figure 1.3a, the static member function is called by the class name in main

**[2]** Abstract base classes and concrete derived classes

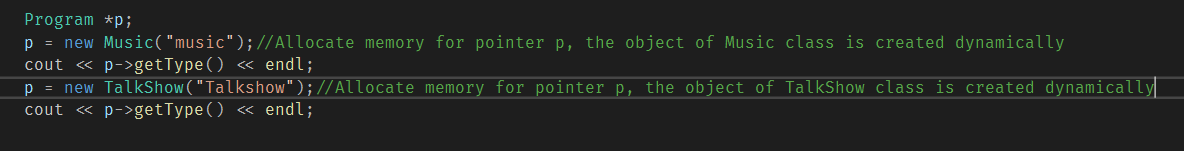
****

**Figure 1.1b abstract base classes**

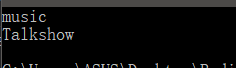


****

**Figure 1.2b concrete derived classes**

****

**Figure 1.3b test code**



**results**

**(b)**

In Figure 1.2b, ‘public:Program’ indicates the two classes inherit from the Program class. The two child classes have additional attribute ‘type’. The pure virtual functions in the Program class indicate this class is an abstract base class. The Music class and TalkShow class both inherit from the abstract base class, they have no pure virtual functions and they give the specific definition for pure virtual functions, so they are concrete derived classes. When the objects of Music class and TalkShow class are created dynamically in main, they call the constructor of their own.

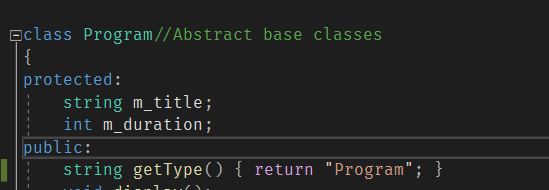
**(c)**

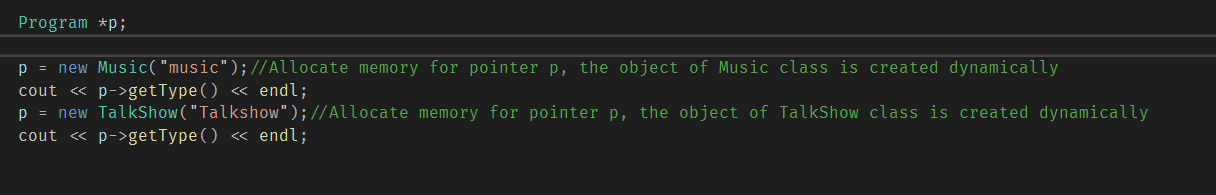
The abstract base classes are the classes that have pure virtual functions, and the concrete derived classes are the classes that inherit from abstract base classes, they have no pure virtual functions and must give specific definitions to the pure virtual functions in abstract base classes. In the example, Music and TalkShow classes have all the attributes, name and duration in the Program class, and they have an additional attribute, type. Their relationship is inheritance. Then, the Program class has pure virtual functions and these functions are given specific definitions in the Music class and TalkShow class. Therefore, this is an suitable example for abstract base classes and concrete derived classes.

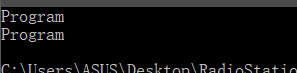
**[3]** Inheritance and polymorphism.

**(a)**

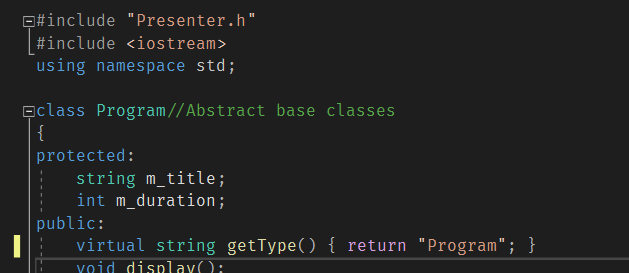
Non-polymorphism:

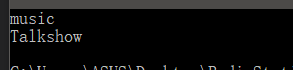






Polymorphism:





**(b)**

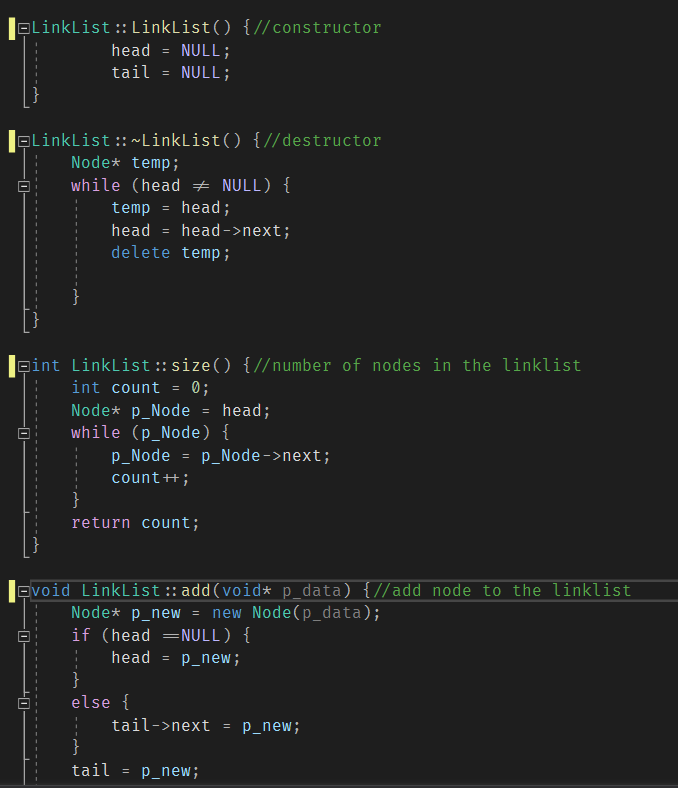
Using virtual functions in parent class can give polymorphism behaviour. When there is polymorphism behaviour, the program will determine which function to call depending on the object type the pointer points to. If there is no polymorphism behaviour, the program always calls the function of the pointer type.

**(c)**

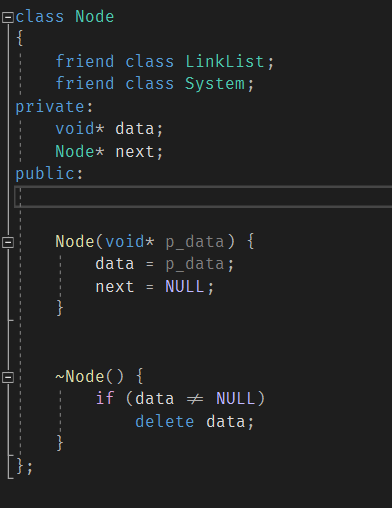
Inheritance is the relationship that child classes inherit all the attributes expect the private ones in the parent class and they should have their own additional attributes. The polymorphism behaviour must be based on the inheritance relationship and making the functions in parent class virtual. It means that when calling a member function, the program will call different functions based on the object type. This is because program determines which function to call at run time if polymorphism behaviour, otherwise, it will determine at compile time. The example is suitable because the relationship is inheritance ,and the parent class and the child classes have the same functions with different definition.

**[4]** Queue link list

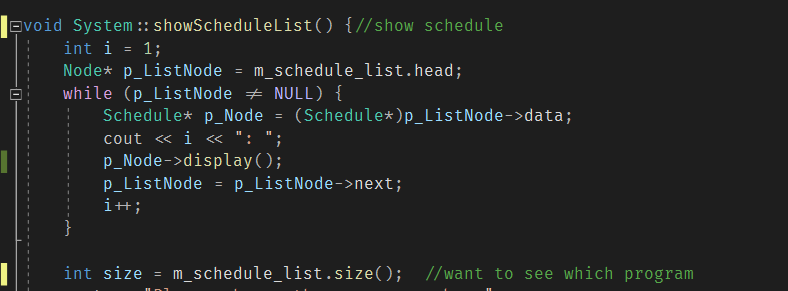
**(a)**

****

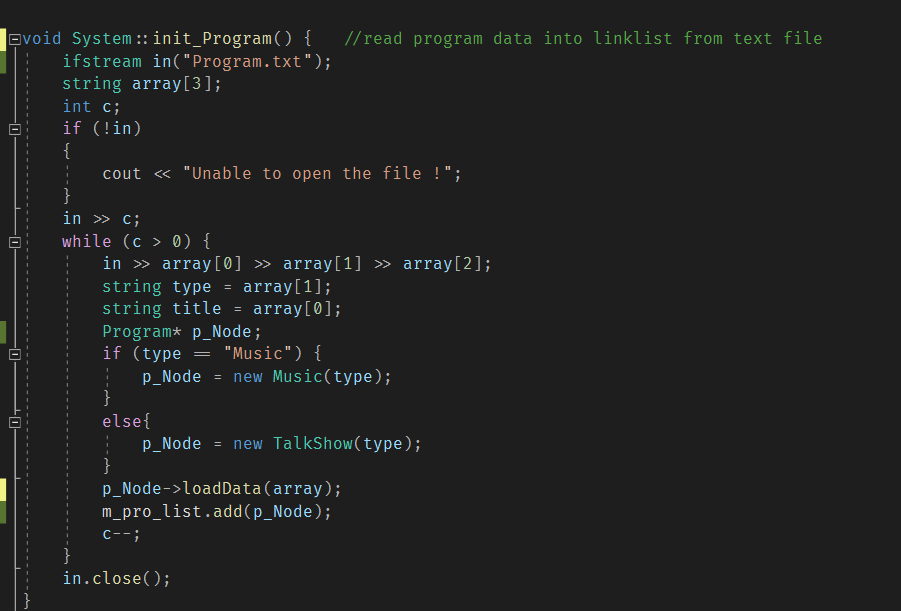
**Figure 4.1a**

****

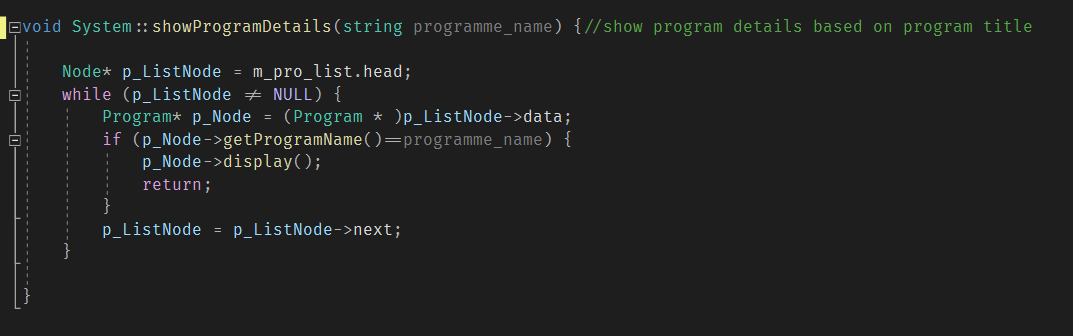
**Figure 4.2a**

****

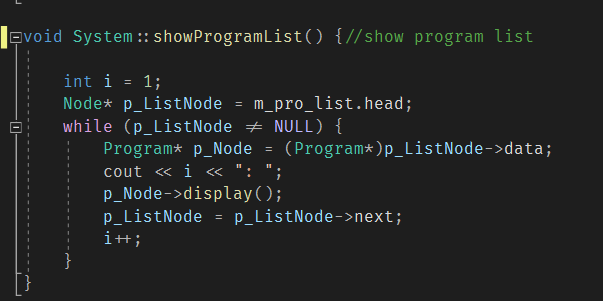
**Figure 4.3a**

****

**Figure 4.4a**



**Figure 4.5a**



**Figure 4.6a**

**(b)**

In Figure 4.2a, in Node class, the data is created as a pointer of void type, so this pointer can point to any types of data. A void pointer should get passed into the constructor and the constructor is called in the add function in LinkList class. in the destructor of Node class, the objects that pointer ‘data’ points to should be deleted to free memory.

In Figure 4.1a, in the constructor of LinkList class, it should make two pointers point to NULL to initialize the link list. Similarly, when a link list destroys, program should delete the objects that pointers point to, in order to free memory.

The Size function in LinkList class is used to count how many data are there in a link list. The function creates a pointer equalling to head pointer so that the operation in this function will not affect the original link list, ‘while(p\_Node)’ means p\_Node not points to NULL, this function returns an integer value and it is called in showSchedule function by link list m\_schedule\_list in Figure 4.3a.

The add function is actually push function as known and it passes into a void\* type value. If head equals to NULL, that indicates there is no data in the link list at all, so the head and tail should both point to the input data., otherwise, the data should be added after the tail of the link list. In Figure 4.4a, add function is called by link list m\_pro\_list.

The init\_program function in Figure 4.4a is used to read program data from text file into link list. The function first read the data into an array of string type, then the function will determine what is the object that p\_Node points to based on the type of program. After that, p\_Node calls loadData function to save the data in array into the object that p\_Node points to, finally, the link list calls add function to add data into the link list.

The showProgramDetails function in Figure 4.5a is used to display specific program. A string value gets passed into the function, the function search the program link list m\_pro\_list to find which data in node has the same program name as the passed value and display the data. Because the data in Node is created as void\*, in Figure 4.5a, when the function want to get the data in Node, ‘(Program\*)’ should be put before p\_ListNode, so the data in Node is changed to Program\* type.

The showProgramList function is similarly to showProgramDetails function, it does not to distinguish and just display all the data in the link list.

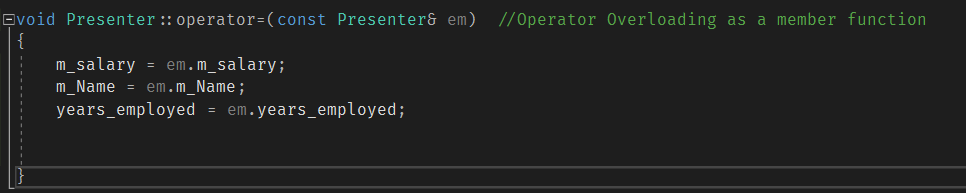
The functions of other two link lists are similar to the program link list.

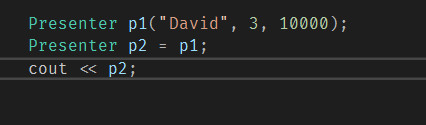
**(c)**

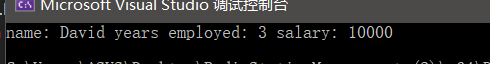
The link list consists of nodes, the address of the next node should be stored in the previous node, so the nodes can be linked up to a link list. Each note in the link list is allocated memory, so when a link list destroys, the addresses that nodes point to should be deleted to free the memory. This project uses three link lists about program, presenter and schedule in System class.

**[5]** Operator Overloading as a member function

**(a)**

****

****



**(b)**

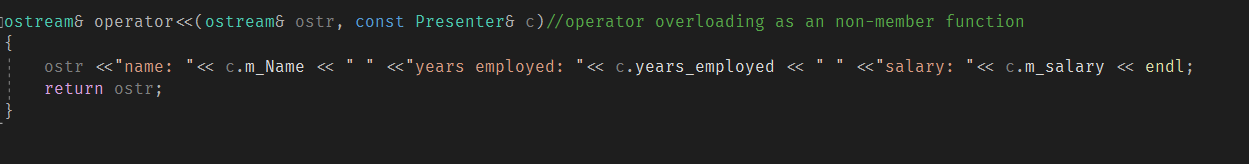
The function is a ‘=’ operator overloading of Presenter class, it is called by the object p2 in main and p1 gets passed into the function. Usually the value at the right of = is unchanged, so the local variable in the function is defined as const. In the function, the values of the attributes of the object passed in is given to the attributes of the object calls this function.

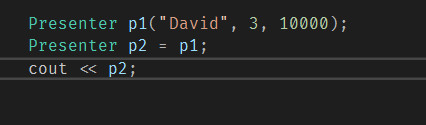
**(c)**

The operator overloading as a member function should be called by the objects of the class where define this function. It is called when the value passed into the function is the same type as the local variable in the function. In the example, operator overloading function is defined in the Presenter class, so just the objects of Presenter class can call this function and the right value should be the type of Presenter.

**[6]** Operator Overloading as a non-member function

**(a)**

****

****

**(b)**

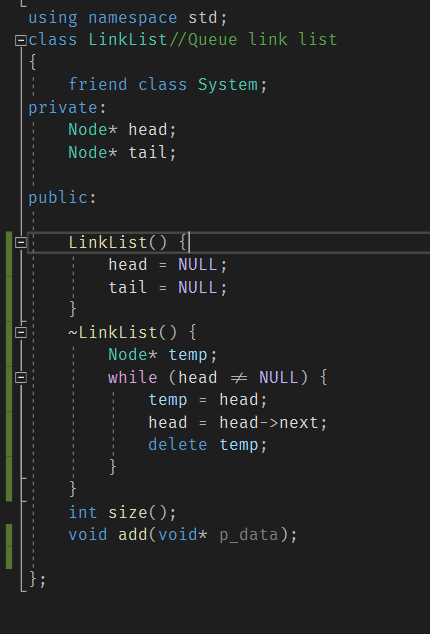
The function is an output operator overloading, it is a friend function of Presenter class. A ostream type value and a Presenter type value should get passed into this function. This function also returns an ostream type value so that this function can be called continually If a string output is inserted. In the function, it outputs every attribute of the Presenter class. This function is called when the output is the object of Presenter class. and in the example, cout and p2 get passed into the function.

**(c)**

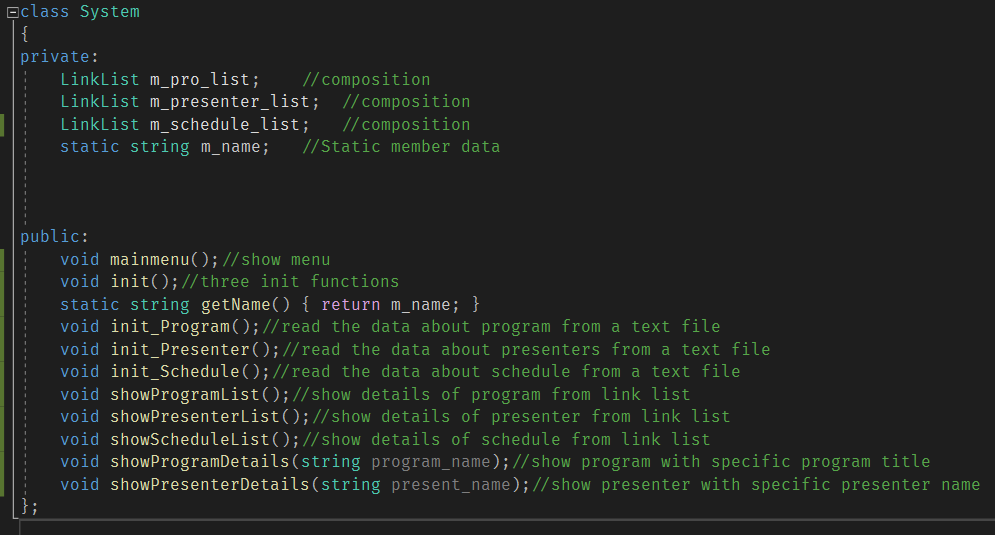
operator overloading as non-member function should be defined as an friend function of a class, so the function can access the private attributes of the class.in the example, operator<< function is defined as a friend function of the Presenter class, so the private attributes in the class can be used in the function.

**[7]** Composition

**(a)**

****

**Figure 7.1a**

****

**Figure 7.2a**

**(b)**

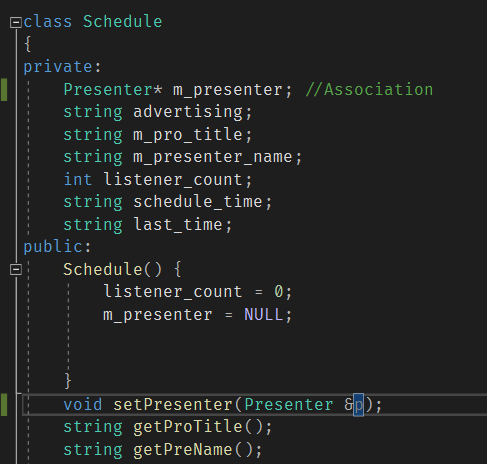
The relationship between Node class and LinkList class is composition and the relationship between LinkList class and System class is also composition. ‘LinkList m\_pro\_list, LinkList…, LinkList …’ makes the composition between LinkList class and System class. ‘delete temp’ makes the composition relationship between Node class and LinkList class.

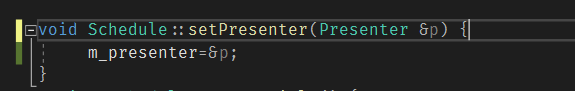
**(c)**

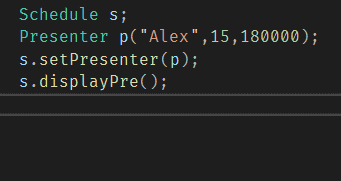
Composition is the relationship between the whole and the part. The whole class includes the member class and they should have the same lifecycle. In Figure 7.1a, LinkList class contains the Node class, through head and tail are created as pointer , in the destructor, the addresses in the two pointers are deleted when the LinkList class destroys, that means these two pointers have the same lifecycle as LinkList class, so their relationship is composition. In Figure 7.2a, the member variables of the LinkList class are created as objects, so they have the same lifecycle. Also, the System class contains the LinkList class. Therefore, their relationship is composition.

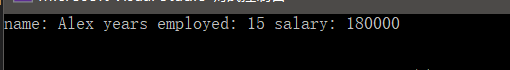
**[8]** Basic Association

**(a)**

****

****



****

**(b)**

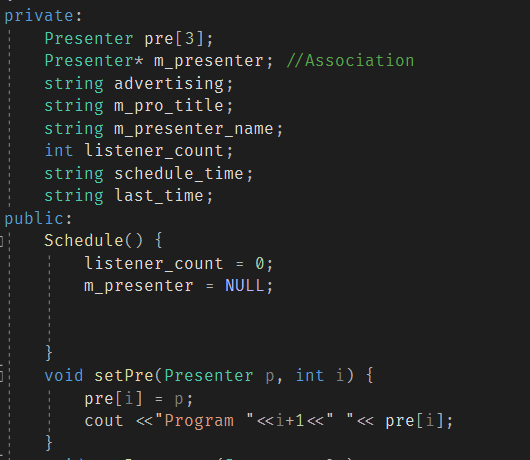
The Presenter class and the Schedule class have the basic association relationship. The two classes are equal, and ‘m\_present’ is created as a pointer, so they have different lifecycle. The function setPresenter gives the address of local variable ‘p’ to member pointer ‘m\_presenter’.

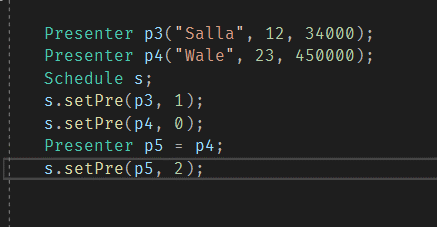
In the example, an object of the Presenter class ‘p’ is created first and call the constructor at the same time, then setPresenter function is called by the object ‘s’ and ‘p’ gets passed into the function. The displayPre function is called by ‘s’ to show the information of presenter in schedule.

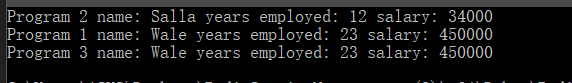
**(c)**

Basic association is the one to one relationship between two equal classes, these two classes have different lifecycle. In the example, it manifested as Schedule arrange the Presenter, which is a one-way association.

**[9]** Qualifier / Qualified Association







**(b)**

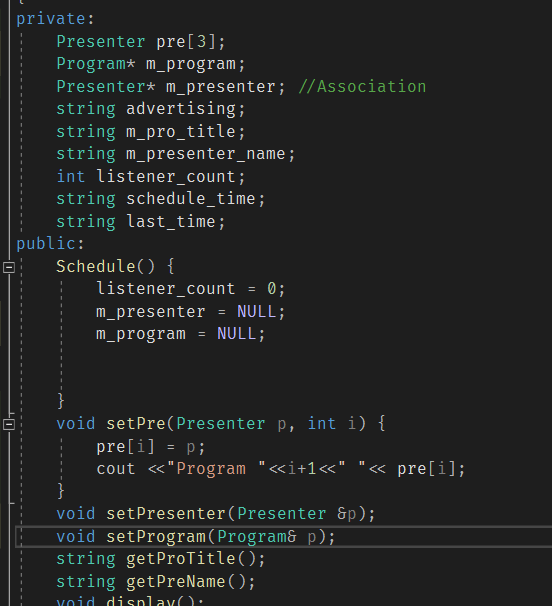
‘pre’ is created as an array of Presenter type in Schedule class, their relationship is Qualifier association, because these are two equal relationship. The setPre function is used to give the Presenter value to a specific node in the array. ‘p3’ and ‘p4’ are created in main calling the constructor of Presenter class. then, setPre function is called by the object s and ‘p3’, ‘1’ get passed in the function, that means give the value of p3 to the second node of array Pre in ‘s’.

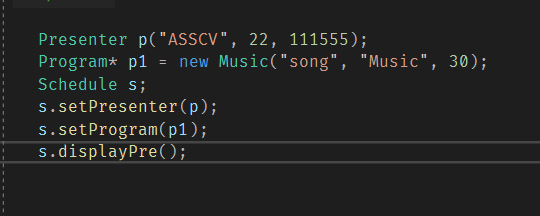
**(c)**

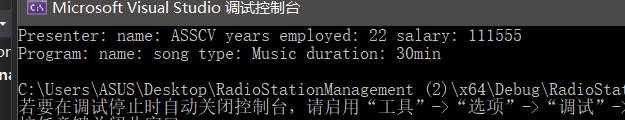
Qualifier or Qualified association is the one to many relationship between two equal classes. In the example, the Presenter class has multiplicity in the Schedule class.

**[10] Association class**

**(a)**

****

****

****

**(b)**

The Schedule class is an association class. Programs should have presenters and presenters need host programs, in Schedule class, presenters can be associated with programs and the corresponding programs of presenters or corresponding presenters of programs can be changed.

**(c)**

Association class is the class allows you to add attributes, operations, and other functions to associations. In the example, the Schedule class can change the arrangement of presenters and programs, so it is a association class.