Classes in C++

Lecture-2

Declaring and defining a class

- class classname {

  access-specifier

  data and functions

  };
- Access-specifier can be

  public

  private

  protected

Note: - by default functions and data declared are private

Access-specifier *(example)*

- #include <iostream>

#include <cstring>

using namespace std;

class employee

{ // class begins

char name[80];

public: void putname(char *); void getname(char *) ;

private: double wage;

public: void putwage(double w); double getwage();

}; // class ends here

```
void employee::putname(char *n)
{ strcpy(name,n); }
void employee::getname(char *n)
{ strcpy(n,name); }
void employee::putwage(double w)
{ wage=w;}
double employee::getwage()
{ return wage; }

int main() { employee ted; char name[80];
ted.putname("Ted Jones"); ted.putwage(7500);
ted.getname(name);
cout<<name<<" make $"<<ted.getwage()<<" per month." ;

return 0;
} // main closing
```

Memory allocation of members

- While objects conceptually contain data members and functions, C++ objects typically contain only data.

- The compiler creates only one copy of the class's member functions and shares that copy amongst all the members.

Placing a class in a separate file for reusability

- benefits of creating class – reusability
- example – C++ standard offers many classes which can be used by including header files
- How to make our classes reusable?

Placing a class in a separate file for reusability

- The program file where the class is declared and defined should not have main() function

- Divide the source code into two parts
  - A .cpp file having main() *(driver program)*
  - A .h file having class declaration

employee.h

```
#include <iostream>
#include <cstring>
using namespace std;
class employee
{  // class begins
char name[80];
public: void putname(char *); void getname(char *) ;
private: double wage;
public: void putwage(double w); double getwage();
}; // class ends here
void employee::putname(char *n)
{ strcpy(name,n); }
void employee::getname(char *n)
{ strcpy(n,name); }
void employee::putwage(double w)
{ wage=w;}
double employee::getwage()
{ return wage; }
```

program1.cpp

```
#include <iostream>
#include “employee.h”
int main()
{ employee ted; char
    name[80];
ted.putname(“Ted
    Jones”);
    ted.putwage(7500);
ted.getname(name);
cout<<name<<“ make
    $”<<ted.getwage()<
    <“ per month.” ;

return 0;
} // main closing
```

Preprocessor directive

- The preprocessor directive

  #include “employee.h”
- Instructs the C++ preprocessor to replace the directive with a copy of the contents of employee.h
- employee.h thus becomes ***re-usable***

How header files are located?

- Used “ “ instead of < >
- “ “ preprocessor locates first in current directory
- < > preprocessor locates in standard directory

Problem

- The abstraction problem is partially solved as placing a class definition in a header file still reveals the entire implementation of the class as the employee.h is a simple text file

Ideal situation

- To use an object of a class, client code should know
  - What member function to call?
  - What arguments to provide?
  - What return type to expect?

Separating interface from implementation

- Interface define and standardize the ways in which things such as people and systems interact with one another.

- The interface of a class describes ***what*** services a class's clients can use and ***how*** to request those services, but not how the class carries out the services.

Separating interface from implementation

- Define member functions outside the class definition, so that their implementation details can be hidden from the client code
- Divide the source code into two parts
  - A .cpp file having main() *(driver program)*
  - A .h file having class definition
  - A .cpp file having member function definition

employee.h

```
#include <iostream>
#include <cstring>
using namespace std;
class employee
{  // class begins
char name[80];
public: void putname(char *); void getname(char *) ;
private: double wage;
public: void putwage(double w); double getwage();
}; // class ends here
```

employee.cpp

```
#include “employee.h”
void employee::putname(char *n)
{ strcpy(name,n); }
void employee::getname(char *n)
{ strcpy(n,name); }
void employee::putwage(double w)
{ wage=w;}
double employee::getwage() { return wage; }
```

program1.cpp

```
#include <iostream>
#include “employee.h”
int main()
{ employee ted; char
    name[80];
ted.putname(“Ted
    Jones”);
    ted.putwage(7500);
ted.getname(name);
cout<<name<<“ make
    $”<<ted.getwage()<
    <“ per month.” ;


return 0;
} // main closing
```

Compilation and Linking process

- Class's interface and implementation will be created and complied by one programmer and used by a separate programmer who implements the class's client code
- Client is provided with the employee.h and the object code of employee.cpp (***not the source file***)

ASSIGNMENT

- What do you mean by Preprocessor directives. Also explain Header and library files.