Previous lecture

We discussed about functions:

- IO from files
- Streams as parameters

Structures and classes

Today:

- Functions in structs
- Data hiding
- Classes and objects
- Inline functions

http://www.icce.rug.nl/documents/cplusplus/cplusplus03.html#l33

Functions in struct

Example:

```
struct Person {
  string name;
  string address;
  void print();
};

void Person::print() { //definition of the member function
   cout << "Name: " << name <<'\n'
   "Address: " <<address<< '\n';
}

int main(){  //example of using Person objects
    Person niece;              //no typedef needed
    niece.name= "Naly James";
    niece.address= "34 Marua Rd, Auckland";
    niece.print();             // use of 'dot' notation
}
```

Notice the difference!

Functions in `struct`

a) The members of a class do not need to use qualified names when referring to the other members of the same class.

b) `typedef` is not needed when declaring variables of struct type in C++.

c) A function that is part of a structure may be selected using the dot (.) (the arrow (->) operator is used when pointers to objects are available).

d) Some advantages of having member functions:

i) the called function automatically accesses the data fields of the structure for which it is invoked;
ii) several types may contain functions having identical names.

Data hiding

Data hiding is the capability of sections of a program to hide its data from other sections.

Two keywords that are related to data hiding are

- private and
- public
- protected-discussed later with inheritance.

These keywords can be used in the definition of structs.

The keyword **public** allows all subsequent fields of a structure to be accessed by all code;
The keyword **private** only allows code that is part of the struct itself to access subsequent fields.

In a struct all fields are public, unless explicitly stated otherwise.

Data hiding

```
struct Person {
public:
   void setName(string n);
   void setAddress(string a);
   void print();
   string getName();
   string getAddress();
private:
   string name;
   string address;
};
```

Available to the outside world

Available only to members in Person struct

Client code:

```
int main(){
   Person std;
   std.setName("Bob");          // OK, setName is public
   std.name= "Karl";  // error, std.name is private
}
```

Data hiding

```
struct Person {
public:
   void setName(string n);
   void setAddress(string a);
   void print();
   string getName();
   string getAddress();
private:
   string  name;
   string address;
};
```

Data hiding benefits:

- **clean data definitions.**
- **enforce their data integrity**

Definition of the member `setName`:

```
void Person::setName(string n){
   name = n;
}
```

Class

Classes are used instead of struct when data hiding is involved.

A class is a kind of struct, except that a class uses **private** access by default, whereas structs use **public** access by default.

```
struct Person {//default public
  void setName(string n);
  void setAddress(string a);
  void print();
  string getName();
  string getAddress();
private:
  string  name;
  string address;
};
```

```
class Person { //default private
  string  name;
  string address;
public:
  void setName(string n);
  void setAddress(string a);
  void print();
  string getName();
  string getAddress();
};
```

C++ struct versus C struct

C approach

```
/* definition of PERSON type. This is C */
typedef struct{
  char name[80];
  char address[80];
} PERSON;

/* initialize fields with a name and address*/
void init(PERSON *p, char const *nm,
          char const *adr);
/* print information */
void print(PERSON const *p);
```

Using the PPERSON type:

```
PERSON son;
init(&son, "Tom", "Napier");
print(&son);
```

C++ approach

```
// Definition of Person type. This is C++
class Person {
public:
  void init(string nm, string adr);
  void print();
private:
   string name;
   string address;
};
```

Using the Person type:

```
Person son;
son.init("Tom", "Napier");
son.print();
```

Defining members

```
class Person {
public:
  void setName(string n){
    name=n;
  }
  void setAddress(string a);
  void print();
  string getName();
  string getAddress();
private:
  string  name;
  string address;
};
```

Inline function

- similar to a #define macro
- might make the code faster
- or totally ignored

http://en.wikipedia.org/wiki/Inline_function

https://isocpp.org/wiki/faq/inline-functions

Defining members

```
class Person {
public:
  void setName(string n);
  void setAddress(string a);
  void print();
  string getName();
  string getAddress();
private:
  string  name;
  string address;
};
```

Best practice for inline members

```
  inline void Person::setName(string n){
    name = n;
  }
```

Inline or not inline?

Inline functions may not be as valuable as they appear:

- The compiler may not be able to inline as many functions as the programmer indicates.
- **The code (of the inline function) gets exposed to its client (the calling function).**
- Inline functions can increase compilation time.
- Inline functions can make the class interface difficult to read/understand.
- Function using recursions and iteration will not be inlined.

Defining members

```
class Person {
public:
  void setName(string n);
  void setAddress(string a);
  void print();
  string getName();
  string getAddress();
private:
  string  name;
  string address;
};
```

Not-inline

```
 void Person::setName(string n){
   name = n;
 }
```

Defining members

```
class Person {
public:
 void setName(string n = "None");
 void setAddress(string a);
 void print();
 string getName()
    {return name;}
 string getAddress()
    {return address;}
private:
  string  name;
  string address;
};
```

Modifier

Accessor

```
void Person::setName(string n){
    name=n;
  }
  void Person::setAddress(string a){
    address=a;
  }
  void Person:: print(){
     cout<<"Name: "<<name<<'\n';
     cout<<"Address: "<<address<<'\n';
  }
```

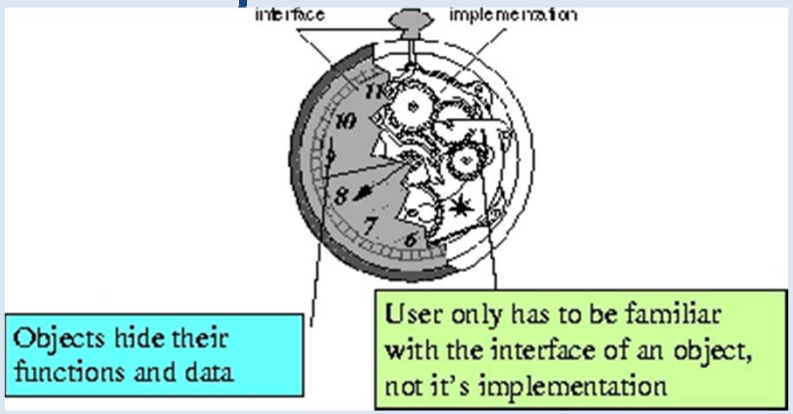
Objects

```
class Person {
public:
 void setName(string n = "None");
 void setAddress(string a);
 void print();
 string getName();
 string getAddress();
private:
   string  name;
   string address;
};
```

Objects

```
int main(){
   Person p,q;
   p.setName("Tom");
   p.setAddress("London, UK");
   Person &a = p;
   a.setName("Eve");
   p.print();
}
```

Encapsulation



- The class interface – public methods the object can execute.
- The class implementation may remain hidden.
- The user only has to be familiar with the interface of an object.

In object oriented languages **type safety** is usually intrinsic in the fact a type system is in place. This is expressed in terms of class definitions.

Type safety is a matter of good class definition: public methods that modify the internal state of an object should preserve the object integrity.

Summary

C++ struct and class notions are very similar;
they differ on default member visibility.

Data hiding is an important mechanism in C++.
Encapsulation is enforcing type safety.

Next

Common member functions