C++ Programming: Session 4

Anupama Chandrasekhar (Aug 23 2018)

Classes and structs

- A **`class/struct`** is a user defined type consisting of data members and member functions
- Member functions can define the meaning of initialization, copy, move and cleanup
- Members are accessed by -> for pointers and . for objects
- Operators can be defined for a **`class`**: +, [], * etc
- A **`class`**'s public members define its interface and private members its implementation
- A **`struct`** is a **`class`** where members are default **`public`**

Anatomy of a class

```
class X {
private:                            // the representation (implementation) is private
     int m;
public:                             // the user interface is public
     X(int i =0) :m{i} { }          // a constructor (initialize the data member m)
     int mf(int i)                  // a member function
     {
          int old = m;
          m = i;                    // set a new value
          return old;               // return the old value
     }
};

X var {7};  // a variable of type X, initialized to 7

int user(X var, X* ptr)
{
    int x = var.mf(7);              // access using . (dot)
    int y = ptr->mf(9);             // access using -> (arrow)
    int z = var.m;                  // error: cannot access private member
}
```

CStash : Motivation for `class`

- CStash is a array like structure whose size can be established at runtime.
- **initialize() :** Performs the initial setup
- **add():** Inserts an element
- **fetch():** Get the element at the index
- **count():** Number of elements in the stash
- **cleanup():** Delete the storage/ any other cleanup tasks.

Dynamic Storage Allocation

- When you do not know the maximum amount of memory that you might need at compile time, you can use get the required memory at runtime from the heap. Heap is a block of memory used for allocating smaller pieces of memory at runtime.
- Dynamic Memory Allocation in C: **`malloc, calloc, realloc`** and **`free`**
- Dynamic Memory Allocation in C++: **`new`** and **`delete`**

```
//General form of new expression

Type* var = new Type;
```

Dynamic Storage Allocation

- Anytime you request a memory, it is possible for the request to fail if there is no more memory
- If **`new`** does not have a corresponding **`delete`** we have a memory leak. Also note we have a special syntax for deleting an array **`delete[]`**.
- Stack variables on the other hand are automatically created and freed by the compiler as the compiler know exactly how much storage is needed and what the scope of the variables are.

Some problems with CStash

- The user of this library should remember to call initialize and cleanup
- Also, if the user of the library decided to initialize the variables themselves, the approach is error prone and there is no way to prevent it.
- Also note, all users of Cstash structure need to include Cstash.h
- Issues with this library:
  - Need to pass the address of the structure to every function
  - Also though we have functions that operate on **Cstash** objects, they have general names like initialize that could lead to name clashes. So you might rename it to something like **Cstash_initialize**, but we can do better…

Solution : Member Functions

- C++ functions can be placed inside **`struct`**s
- You no longer need to pass the address of the **`struct`** as the first argument (it is done implicitly by the compiler, so it the name decoration)
- You can still use :: to refer to the member function
- Member selection is gone, you can directly use the member variables, the compiler secretly passes the address of the object.
- If you want to get hold of the address of the object, you can use the this pointer.
- Also note that automatic casting doesn't happen in C++ and we need to be explicit about it.
- Also notice that having member functions be the only one accessing data members can make debugging easier.

What is an object?

- Combining data and the functions together in a **`struct`** and thus enabling a new way of thinking about structures as more than a mere collection of different data types is the essence of the programming paradigm shift that C++ brings to C. So now you can have a user defined datatype with characteristics and behaviors.
- A C++ object is just a variable of a user defined type. It allows us to abstract a concept. Also with a new type defined we have all of C++'s type checking mechanisms applied to the objects, meaning a function that can accept only a specific user defined type will not accept another type.

More about objects

- How big is an object? .. About the same size as you would expect from a struct. You can determine the size of a **`struct`** with the **`sizeof()`** operator
- Each object must have a unique address, an interesting side effect is that structures with no data members will always have some minimum non zero size.

Header files

- Place all **`struct`** and file declarations in a header file and include that header file with the uses.
- Don’t do anything in a header that will cause ambiguity at link time.
- You can prevent multiple definitions with **`#ifndef`** guards

```
#ifndef FOO
#define FOO

struct bar {
  int  a;
  char b;
  void initialize()
  {
    a = 0;
    b= '0';
  };
};
#endif //FOO
```

Access Modifiers

- C **`struct`**s have no boundaries. There is no way to enforce **initialize()** or **cleanup()** and there is no way to hide some members of the **`struct`**.
- The reason to control access is:
  - Don’t allow the user to do unsafe things
  - Separate implementation from the interface
- C++ achieves this separation through access modifiers: **`public`**, **`private`** and **`protected`**
- There is no required order for access specifiers, they may appear more than once and affect all the members after them until the next access specifier. Exception: **`friend`**

Access Control

- Access control is often called *implementation hiding* and is one of the key ideas of object oriented programming. And this is the important addition to C++ differentiating it from C, enough that the idea warranted a new keyword **`class`**.
- **`class`** defaults to private while **`struct`** defaults to public.
- Modify Stash to use access control <see example>

PIMPL Design Pattern

- See example **Handle.cpp**

Problem for next Session

- Design a Parking Lot

Credits

- Bruce Eckel: Thinking in C++ for all the code examples!