Lec-5: C++ Structures & Classes

Bernard Nongpoh

Structs

Why Structs

```
string city1 = "Guwahati";
string addressLine1 = "Department of CSE";
int pincode1 = 781039;
```

Why Structs

```
string city1 = "Guwahati";
string addressLine1 = "Department of CSE";
int pincode1 = 781039;
```

```
string city2 = "Guwahati";
string addressLine2 = "Department of EEE";
int pincode2 = 781039;
```

Why Structs

```
string city1 = "Guwahati";
string addressLine1 = "Department of CSE";
int pincode1 = 781039;
```

```
string city2 = "Guwahati";
string addressLine2 = "Department of EEE";
int pincode2 = 781039;
```

```
std::cout<<city1<<std::endl;
std::cout<<addressLine1<<std::endl;
std::cout<<pincode1<<std::endl;
```

```
std::cout<<city2<<std::endl;
std::cout<<addressLine2<<std::endl;
std::cout<<pincode12<<std::endl;
```

Why Structs

```
string city1 = "Guwahati";
std::cout<<pincode12<<std::endl;
```

```
void printAddress(string city, string addressLine, int pincode){
std::cout<<city<<std::endl;
std::cout<<addressLine<<std::endl;
std::cout<<pincode<<std::endl;
}

int main(){
  string city1 = "Guwahati";
  string addressLine1 = "Department of CSE";
  int pincode1 = 781039;
  printAddress(city1, addressLine1, pincode1);
  return 0;
}
```

Why Structs

```
string city1 = "Guwahati";
std::cout<<pincode2<<std::endl;
```

```
void printAddress(string city, string addressLine, int pincode){
std::cout<<
std::cout<<
std::cout<<
}

int main(){
  string ci
  string ad
  int pincode1 = 781039;
  printAddress(city1, addressLine1, pincode1);
  return 0;
}
```

Issues:

- Data is not grouped
- Hard to manage many addresses
- Code becomes messy
- Cannot pass address as one unit

Why Structs

```
return_type getAddress(){
// return city, addressLine, pincode
// How can we return all three things?
// What should our return type be?
}
```

Why Structs

```
return_type getAddress(){
// return city, addressLine, pincode
// How can we return all three things?
// What should our return type be?
}
```

- A **`struct`** is a user-defined data type in C++ that groups different related variables into one single unit

Structs bundle multiple data types together.

```
struct Address {
    string city; // These are called fields
    string addressLine; // Each has a name and type
    int pincode;
};

Address address1; // Initialize struct
address1.city = “Guwahati”;
address1.addressLine = “Department of CSE”; // Access filed with ‘.’
address1.pincode = 781039;
```

[Source]

Returning multiple values

```
Address getAddress(){

  Address address1;

  address1.city = “Guwahati”;
  address1.addressLine = “Department of CSE”;
  address1.pincode = 781039;

return address1;
}
```

[Source]

Structs Initialization

```
struct Address {
    string city; // These are called fields
    string addressLine; // Each has a name and type
    int pincode;
};

Address addressCSE = {“Guwahati”,“Department of CSE”, 781039  }
// Order depends on field order in struct. = is optional
Address addressEEE{“Guwahati”,“Department of EEE”, 781039  }
```

[Source]

Structs Initialization

```
struct Address {
    string city; // These are called fields
    string addressLine; // Each has a name and type
    int pincode;
};

Address addressCSE = {“Guwahati”,“Department of CSE”, 781039  }
// Order depends on field order in struct. = is optional
Address addressEEE{“Guwahati”,“Department of EEE”, 781039  }
```

A struct bundles named variables into a new type

[Source]

Many Possible Structs

```
struct Name {
    string first;
    string last;
};

Name rf = { "Rachel", "Fernandez" };
```

[Source]

Many Possible Structs

```
struct Name {
    string first;
    string last;
};

Name rf = { "Rachel", "Fernandez" };
```

```
struct Order {
    string item;
    int quantity;
};

Order dozen = { "Eggs", 12 };
```

[Source]

Many Possible Structs

```
struct Name {
    string first;
    string last;
};

Name rf = { "Rachel", "Fernandez" };
```

```
struct Order {
    string item;
    int quantity;
};

Order dozen = { "Eggs", 12 };
```

```
struct Point {
    double x;
    double y;
};

Point origin { 0.0, 0.0 };
```

[Source]

Many Possible Structs

```
struct Name {
    string first;
    string last;
};

Name rf = { "Rachel", "Fernandez" };
```

```
struct Order {
    string item;
    int quantity;
};

Order dozen = { "Eggs", 12 };
```

Notice anything?

```
struct Point {
    double x;
    double y;
};

Point origin { 0.0, 0.0 };
```

```
struct Circle {
    Point center;
    double radius;
};

Circle circle { {0, 0} , 50000000 };
```

[Source]

Example: Shop Billing System

Shop Billing System

- **Shop Billing System** that reads an item's **name** and **pricing details** from the user.

Each item should store:

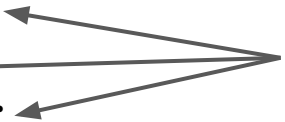
- Base price
- Discount %
- GST %
- Calculate the **final payable amount** after applying **discount first, then GST**, and display the bill.

Example: Shop Billing System

```
    struct Item {
    string name;
    float basePrice;
    float discount;
    float gstPercent;
};
```

Example: Shop Billing System

```
    struct Item {
    string name;
    float basePrice;
    float discount;
    float gstPercent;
};
```



Can I group these 3 fields into Pricing?

Example: Shop Billing System

```
    struct Pricing {
        float basePrice;
        float discount;
        float gstPercent;
    };

    struct Item {
    string name;
    Pricing cost;
};
```

Example: Shop Billing System

```
    struct Pricing {
        float basePrice;
        float discount;
        float gstPercent;
    };

    struct Item {
    string name;
    Pricing cost;
};
```

```
Item item;
item.name =”Pen”;
item.cost.basePrice = 10.5;
item.cost.discount = 5;
item.cost.gstPercent = 18
```

```
float priceAfterDiscount = item.cost.basePrice-(item.cost.basePrice * item.cost.discount/100);
float finalPrice = priceAfterDiscount + (priceAfterDiscount * item.cost.gstPercent / 100);
```

[Code]

Function inside struct

```
struct Pricing {
        float basePrice;
        float discount;
        float gstPercent;
};

struct Item {
    string name;
    Pricing cost;
     // 3. Function inside struct (GST calculation)
    float finalPrice() {
        float priceAfterDiscount = basePrice-(basePrice * discount/100);
        float finalPrice = priceAfterDiscount + (priceAfterDiscount * gstPercent / 100);
        return finalPrice;
}

};
```

[Code]

Function inside struct

```
struct Pricing {
        float basePrice;
        float discount;
        float gstPercent;
};

struct Item {
    string name;
    Pricing cost;
     // 3. Function inside struct (GST calculation)
    float finalPrice() {
        float priceAfterDiscount = basePrice-(basePrice * discount/100);
        float finalPrice = priceAfterDiscount + (priceAfterDiscount * gstPercent / 100);
        Retur n finalPrice;
}

};
```

Struct == Class, but??

[Code]

Any Questions?

C++ Classes

- **C Structure**
  - A C structure (struct) is a collection of variables of the same or different data types under a single name

- **C++ class**
  - A class (class) extends the concept of structure to hold functions as members

Source

Why classes?

- C has no objects
- No way of **encapsulating** data and the functions that operate on that data
- No ability to have object-oriented programming (OOP) design patterns

Source

What is object-oriented-programming?

- Object-oriented-programming is centered around **objects**
- Focuses on design and implementation of classes
- Classes are the **user-defined types** that can be declared as an object

Source

Comparing struct and class (1)

- Structures and classes are semantically equivalent in C++. However, the keywords should be used to distinguish between different semantics:
  - Struct represents passive objects, namely the physical state (set of data)
  - Class represents active objects, namely logical state (data abstraction)

Source

Comparing struct and class (2)

- Structures and classes are semantically equivalent in C++. However, the keywords should be used to distinguish between different semantics:
  - Struct represents passive objects, namely the physical state (set of data)
  - Class represents active objects, namely logical state (data abstraction)
- **Default member access**: In a `struct`, all members are **public by default**, whereas in a `class`, all members are **private by default**
- **Default inheritance access**: When a `struct` inherits from another type, the inheritance is **public by default**, while in a `class`, the inheritance is **private by default**

Source

Comparing struct and class (3)

Choosing `struct` vs `class` is about intent, not capability

Source

Class Members- Data and Function Members

- **Data Member:** data within a `class` are called data members or class fields
- **Function Member:** functions within a `class` are called function members or methods

Source

struct/class Declaration and Definition (1)

struct declaration and definition

```
struct A; // struct declaration
struct A{
int x; // data member
void f();// function member
};
```

class declaration and definition

```
class A; // struct declaration
class A{
int x; // data member
void f();// function member
};
```

Source

struct/class Declaration and Definition (2)

Struct declaration and definition

```
struct A{
void g(); // function member declaration
void f(){ // function member declaration, inline definition
    cout<<"f";
}
};
void A::g(){ // function member declaration, out-of-line definition
cout<<"g";
}
```

Source

struct/class Declaration and Definition (3)

struct declaration and definition

```
struct B{
void g(){ // function member
cout<<"g";
}

struct A{
int x; // data member
B b; // data member
void f(){ // function member
cout<<"f";
}
};

A a;
A.x;
a.f();
a.b.g();
```

Source

Exercise (1)

Now comes the important part: thinking first, code later. Let’s design a class together.

Source

Exercise (2)

A class is a real-world thing that has

- properties (data) and
- behaviors (actions)

Source