ACA Summer School 2014
Advanced C++

Pankaj Prateek

ACA, CSE, IIT Kanpur

June 23, 2014

- Course website:
  `http://www.cse.iitk.ac.in/users/aca/sumschool2014.html`
- Evaluation
  - End Sem - 50%
  - Assignments / In-class quiz - 50%
- Timings:
  - M-F 1430 - 1600 hrs

Prerequisites

- A good command over any programming language preferably C/C++
  - Pointers
  - Structures

Structures

- How do you store the details of a person?

```
char strName[20];
int nBirthYear;
int nBirthMonth;
int nBirthDay;
int nHeight;
```

- Information is not grouped in any way.
- To pass your information to a function, you would have to pass each variable independently.
- If wanted to store information about many people, would have to declare arrays.

Structures

- How do you store the details of a person?

```
char strName[20];
int nBirthYear;
int nBirthMonth;
int nBirthDay;
int nHeight;
```

- Information is not grouped in any way.
- To pass your information to a function, you would have to pass each variable independently.
- If wanted to store information about many people, would have to declare arrays.

Structures

- How do you store the details of a person?

```
char strName[20];
int nBirthYear;
int nBirthMonth;
int nBirthDay;
int nHeight;
```

- Information is not grouped in any way.
- To pass your information to a function, you would have to pass each variable independently.
- If wanted to store information about many people, would have to declare arrays.

Structures

- How do you store the details of a person?

```
char strName[20];
int nBirthYear;
int nBirthMonth;
int nBirthDay;
int nHeight;
```

- Information is not grouped in any way.
- To pass your information to a function, you would have to pass each variable independently.
- If wanted to store information about many people, would have to declare arrays.

Structures

- How do you store the details of a person?

```
char strName[20];
int nBirthYear;
int nBirthMonth;
int nBirthDay;
int nHeight;
```

- Information is not grouped in any way.
- To pass your information to a function, you would have to pass each variable independently.
- If wanted to store information about many people, would have to declare arrays.

Structures

- C++ allows to create own user-defined data types to aggregate different variables : `structs`
- Structure declaration

```
struct Person{
   char strName[20];
   int nBirthYear;
   int nBirthMonth;
   int nBirthDay;
   int nHeight;
}; //DO NOT forget a semicolon
```

- Now person structure can be used as a built-in variable.

Structures

- C++ allows to create own user-defined data types to aggregate different variables : `structs`
- Structure declaration

```
struct Person{
   char strName[20];
   int nBirthYear;
   int nBirthMonth;
   int nBirthDay;
   int nHeight;
}; //DO NOT forget a semicolon
```

- Now `person` structure can be used as a built-in variable.

Structures

- C++ allows to create own user-defined data types to aggregate different variables : `structs`
- Structure declaration

```
struct Person{
  char strName[20];
  int nBirthYear;
  int nBirthMonth;
  int nBirthDay;
  int nHeight;
}; //DO NOT forget a semicolon
```

- Now `person` structure can be used as a built-in variable.

Structures

- Usage

```
struct Person p1;
struct Person p2;
```

- Accessing Members:

```
p1.name = ``pankaj'';
p1.nBirthDay = 20;
p2.name = ``Rahul'';
```

Structures

- Usage

```
struct Person p1;
struct Person p2;
```

- Accessing Members:

```
p1.name = ``pankaj'';
p1.nBirthDay = 20;
p2.name = ``Rahul'';
```

Structures

- No memory is allocated in structure declaration
- Size of a structure is the sum of sizes of it elements
- Can pass entire structures to functions

```
void PrintInfo(struct Person p) {
  cout << ``Name: '' << p.name <<endl;
  cout << ``bDay: '' << p.nBirthDay <<endl;
}
int main() {
  struct Person p1;
  p1.name = ``Pankaj'';
  p1.nBirthDay = 20;
  PrintInfo(p1);
}
```

Structures

- No memory is allocated in structure declaration
- Size of a structure is the sum of sizes of it elements
- Can pass entire structures to functions

```
void PrintInfo(struct Person p) {
   cout << ``Name: '' << p.name <<endl;
   cout << ``bDay: '' << p.nBirthDay <<endl;
}
int main() {
   struct Person p1;
   p1.name = ``Pankaj'';
   p1.nBirthDay = 20;
   PrintInfo(p1);
}
```

Structures

- No memory is allocated in structure declaration
- Size of a structure is the sum of sizes of it elements
- Can pass entire structures to functions

```
void PrintInfo(struct Person p) {
  cout << ``Name: '' << p.name <<endl;
  cout << ``bDay: '' << p.nBirthDay <<endl;
}
int main() {
  struct Person p1;
  p1.name = ``Pankaj'';
  p1.nBirthDay = 20;
  PrintInfo(p1);
}
```

Structures

- Always have to use the word "`struct`"
- No explicit connection between members of a structure and the functions manipulating them
- Cannot be treated as built-in types (c1 + c2 is not valid for instances of "`struct complex`")
- Data hiding is not permitted (Why do we need this?)
- All members of a structure are by defaut "`public`" (will be discussed later)

Structures

- Always have to use the word "`struct`"
- No explicit connection between members of a structure and the functions manipulating them
- Cannot be treated as built-in types (c1 + c2 is not valid for instances of "`struct complex`")
- Data hiding is not permitted (Why do we need this?)
- All members of a structure are by defaut "`public`" (will be discussed later)

Structures

- Always have to use the word "`struct`"
- No explicit connection between members of a structure and the functions manipulating them
- Cannot be treated as built-in types (c1 + c2 is not valid for instances of "`struct complex`")
- Data hiding is not permitted (Why do we need this?)
- All members of a structure are by defaut "`public`" (will be discussed later)

Structures

- Always have to use the word "`struct`"
- No explicit connection between members of a structure and the functions manipulating them
- Cannot be treated as built-in types (c1 + c2 is not valid for instances of "`struct complex`")
- Data hiding is not permitted (Why do we need this?)
- All members of a structure are by defaut "`public`" (will be discussed later)

Structures

- Always have to use the word "`struct`"
- No explicit connection between members of a structure and the functions manipulating them
- Cannot be treated as built-in types (c1 + c2 is not valid for instances of "`struct complex`")
- Data hiding is not permitted (Why do we need this?)
- All members of a structure are by defaut "`public`" (will be discussed later)

Classes

- Extension of structures
- Class Definition

```
class class_name {
private:
   variable declarations;
   function declarations;
public:
   variable declarations;
   function declarations;
}; //DO NOT forget the semicolon
```

Classes

- Extension of structures
- Class Definition

```
class class_name {
private:
   variable declarations;
   function declarations;
public:
   variable declarations;
   function declarations;
}; //DO NOT forget the semicolon
```

Class: Example

- Example

```
class item {
private:
   int number;       // variable declarations
   float cost;       // private by default!
public:
   // function declarations through prototypes
   void getData(int a, float b);
   void putData(void);
}
```

Class

No memory allocated for a class definiton. It is only a “template”, like a definition of a structure

Class: Public and Private

- Private members are not directly accessible outside the class. They are "hidden" from the outside world. The only way to access them is by using public functions (if defined) which manipulate them.
- Public members can be accessed using the dot operator (member selection operator). Eg. student.name, item.getData()

Class: Public and Private

- Private members are not directly accessible outside the class. They are "hidden" from the outside world. The only way to access them is by using public functions (if defined) which manipulate them.
- Public members can be accessed using the dot operator (member selection operator). Eg. student.name, item.getData()

Class: Objects

- Class, like a structure definition, is a user-defined data type without any concrete existance.
- A concrete instance of a class is an object.
- Memory is allocated for every object that is created.
- An independent new set of variables is created for each object.
- Public members can be accessed by using the dot operator on the object while private members cannot be accessed directly.

Class: Objects

- Class, like a structure definition, is a user-defined data type without any concrete existance.
- A concrete instance of a class is an object.
- Memory is allocated for every object that is created.
- An independent new set of variables is created for each object.
- Public members can be accessed by using the dot operator on the object while private members cannot be accessed directly.

Class: Objects

- Class, like a structure definition, is a user-defined data type without any concrete existance.
- A concrete instance of a class is an object.
- Memory is allocated for every object that is created.
- An independent new set of variables is created for each object.
- Public members can be accessed by using the dot operator on the object while private members cannot be accessed directly.

Class: Objects

- Class, like a structure definition, is a user-defined data type without any concrete existance.
- A concrete instance of a class is an object.
- Memory is allocated for every object that is created.
- An independent new set of variables is created for each object.
- Public members can be accessed by using the dot operator on the object while private members cannot be accessed directly.

Class: Objects

- Class, like a structure definition, is a user-defined data type without any concrete existance.
- A concrete instance of a class is an object.
- Memory is allocated for every object that is created.
- An independent new set of variables is created for each object.
- Public members can be accessed by using the dot operator on the object while private members cannot be accessed directly.

Class: Objects

Object Creation

- Just like variable declaration
- Memory is allocated for every object that is created
- Example:

```
item a;
item b,c,d;
```

Class: Objects

Object Creation

- Just like variable declaration
- Memory is allocated for every object that is created
- Example:

```
item a;
item b,c,d;
```

Class: Objects

Object Creation

- Just like variable declaration
- Memory is allocated for every object that is created
- Example:

```
item a;
item b,c,d;
```

Class: Objects

Object Creation

- Just like variable declaration
- Memory is allocated for every object that is created
- Example:

```
item a;
item b,c,d;
```

Class: Function definitions

Outside the class

```
class Employee {
  int empno, salary;
public:
  void set(int roll, int sal);
};

void employee::set(int roll, int sal) {
  empno = roll;
  salary = sal;
}
```

Class: Function definitions

Outside the class

```
class Employee {
  int empno, salary;
public:
  void set(int roll, int sal);
};

void employee::set(int roll, int sal) {
  empno = roll;
  salary = sal;
}
```

Class: Function definitions

Inside the class

```
class Employee {
  int empno, salary;
public:
  void set(int roll, int sal) {
    empno = roll;
    salary = sal;
  }
};
```

Class: Function definitions

Inside the class

```
class Employee {
  int empno, salary;
public:
  void set(int roll, int sal) {
    empno = roll;
    salary = sal;
  }
};
```

Important Properties

- Invoking other member functions from inside a member function does not require explicit use of the object
- Array size, if used inside classes, need to be determined at compile time (for dynamic arrays, with size to be determined at run time, "new" operator is used inside a constructor, will be discussed later)
- Arrays of objects are allowed (stored contiguously). Objects can be used as members of some other class in nested fashion.
- Objects, just like built-in types, can be return type of functions.
- Private functions and variables can only be accessed from within the class.

Important Properties

- Invoking other member functions from inside a member function does not require explicit use of the object
- Array size, if used inside classes, need to be determined at compile time (for dynamic arrays, with size to be determined at run time, "new" operator is used inside a constructor, will be discussed later)
- Arrays of objects are allowed (stored contiguously). Objects can be used as members of some other class in nested fashion.
- Objects, just like built-in types, can be return type of functions.
- Private functions and variables can only be accessed from within the class.

Important Properties

- Invoking other member functions from inside a member function does not require explicit use of the object
- Array size, if used inside classes, need to be determined at compile time (for dynamic arrays, with size to be determined at run time, "new" operator is used inside a constructor, will be discussed later)
- Arrays of objects are allowed (stored contiguously). Objects can be used as members of some other class in nested fashion.
- Objects, just like built-in types, can be return type of functions.
- Private functions and variables can only be accessed from within the class.

Important Properties

- Invoking other member functions from inside a member function does not require explicit use of the object
- Array size, if used inside classes, need to be determined at compile time (for dynamic arrays, with size to be determined at run time, "new" operator is used inside a constructor, will be discussed later)
- Arrays of objects are allowed (stored contiguously). Objects can be used as members of some other class in nested fashion.
- Objects, just like built-in types, can be return type of functions.
- Private functions and variables can only be accessed from within the class.

Important Properties

- Invoking other member functions from inside a member function does not require explicit use of the object
- Array size, if used inside classes, need to be determined at compile time (for dynamic arrays, with size to be determined at run time, "new" operator is used inside a constructor, will be discussed later)
- Arrays of objects are allowed (stored contiguously). Objects can be used as members of some other class in nested fashion.
- Objects, just like built-in types, can be return type of functions.
- Private functions and variables can only be accessed from within the class.

Classes

Problem

- Consider a class "car" which has the carNo, carModel, carMake fields and relevant functions to modify them.
- You have to count the number of objects of the class created. How do you do it?

Classes

Problem

- Consider a class “car” which has the carNo, carModel, carMake fields and relevant functions to modify them.
- You have to count the number of objects of the class created. How do you do it?

Classes

Problem

- Consider a class "car" which has the carNo, carModel, carMake fields and relevant functions to modify them.
- You have to count the number of objects of the class created. How do you do it?

Static Members

```
class item {
  static int count;
  // rest of class definition
};
int item::count;
```

Static Members

Properties

- Every static member needs to be defined outside the class as well.
- Only one copy of the static variable is shared among all objects of the class.
- Visible only within the class but exists for the lifetime of the program.
- No object instantiation is required to access static members.

Static Members

Properties

- Every static member needs to be defined outside the class as well.
- Only one copy of the static variable is shared among all objects of the class.
- Visible only within the class but exists for the lifetime of the program.
- No object instantiation is required to access static members.

Static Members

Properties

- Every static member needs to be defined outside the class as well.
- Only one copy of the static variable is shared among all objects of the class.
- Visible only within the class but exists for the lifetime of the program.
- No object instantiation is required to access static members.

Static Members

Properties

- Every static member needs to be defined outside the class as well.
- Only one copy of the static variable is shared among all objects of the class.
- Visible only within the class but exists for the lifetime of the program.
- No object instantiation is required to access static members.

Static Members

Properties

- Every static member needs to be defined outside the class as well.
- Only one copy of the static variable is shared among all objects of the class.
- Visible only within the class but exists for the lifetime of the program.
- No object instantiation is required to access static members.

Memory Allocation of Objects

- For member functions and static variables, memory is allocated when the class is defined, all objects of the class share the same function code and static variables (every class does not get its own copy). "`this`" pointer is implicitly passed so that the function code is executed on the correct class instance.
- For variables, memory is allocated when the object is defined. Each instance gets its own set of variables.

Memory Allocation of Objects

- For member functions and static variables, memory is allocated when the class is defined, all objects of the class share the same function code and static variables (every class does not get its own copy). "`this`" pointer is implicitly passed so that the function code is executed on the correct class instance.
- For variables, memory is allocated when the object is defined. Each instance gets its own set of variables.

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Shared functions among multiple classes
- Properties:
  - Can access private members of the class
  - Often used in operator overloading
  - Not in the scope of the class. Cannot be called using an object of the class.
  - Can be invoked like a normal function
  - Cannot access members of a class directly without an object of the class

Class: Friend

- Properties: (cont. . .)
  - Can be declared as either private or public without changing the meaning
  - Objects can be passed to the function by value or by reference
  - A class can be defined to be the friend of another class. In that case, all member functions of one class are friends of the other class

Class: Friend

- Properties: (cont. . .)
  - Can be declared as either private or public without changing the meaning
  - Objects can be passed to the function by value or by reference
  - A class can be defined to be the friend of another class. In that case, all member functions of one class are friends of the other class

Class: Friend

- Properties: (cont. . .)
  - Can be declared as either private or public without changing the meaning
  - Objects can be passed to the function by value or by reference
  - A class can be defined to be the friend of another class. In that case, all member functions of one class are friends of the other class

Class: Friend

- Properties: (cont. . .)
  - Can be declared as either private or public without changing the meaning
  - Objects can be passed to the function by value or by reference
  - A class can be defined to be the friend of another class. In that case, all member functions of one class are friends of the other class

References

- The C++ Programming Language
  - Bjarne Stroustrup
- Object Oriented Programming with C++
  - E Balaguruswamy
- `http://www.cplusplus.com/reference`
- `http://www.learncpp.com/`
- `http://www.java2s.com/Code/Cpp/CatalogCpp.htm`