Week 10 Lecture 1

NWEN 241
Systems Programming

Alvin Valera

Alvin.valera@ecs.vuw.ac.nz

Content

- Strings
- Inheritance

Strings in C++

Strings in C++

- **C Strings**
  - A one-dimensional array of characters

- Standard C++ Library string class - **std::string**
  - A container for handling char arrays

Recap: C Strings

- A string is a sequence of characters that is terminated by a null character
- In C a string is a one-dimensional array of characters
- There are functions declared to manipulate strings in string.h
- Supported in C++. The `cstring` library can be used.
- Code safety / security is the responsibility of the programmer

`strcpy(s1, s2);` Copies string s2 into string s1.
`strcat(s1, s2);` Concatenates string s2 onto the end of string s1.
`strlen(s1);` Returns the length of string s1.
`strcmp(s1, s2);` Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
`strchr(s1, ch);` Returns a pointer to the first occurrence of character ch in string s1.
`strstr(s1, s2);` Returns a pointer to the first occurrence of string s2 in string s1.

std::string in C++

- C++ has a **`string`** class type that implements string datatype
- Not *exactly similar* to Java String class
  - Java strings are immutable reference types; C++ strings are mutable!
- Wide range of operators and member functions are available for variables declared as string type
- Include **`<string>`** header file
- Use **`standard`** namespace

```
#include <iostream>
#include <string>
using namespace std;

int main (void)
{
  string s1, s2; // empty
  s1 = "Hello";
  s2 = "Hello World !";

  cout << "String 1: " << s1 << "Length: "
  << s1.length() << endl;

  cout << "String 2: " << s2 << “Length: "
  << s2.length() << endl;

  return 0;
}
```

Character Array vs String Class in C++

String Operation	Character Array	String class
Copy string s2 into string s1.	`strcpy(s1, s2);`	`s1 = s2;`
Concatenates string s2 onto the end of string s1.	`strcat(s1, s2);`	`s1 + s2;`
Returns the length of string s1.	`strlen(s1);`	`s1.length();`
Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.	`strcmp(s1, s2);`	`==, <=, >, and >=` operators Or `s1.compare(s2)`
Returns a pointer to the first occurrence of character ch in string s1.	strchr(s1, ch);	`s1.find(ch)`
Returns a pointer to the first occurrence of string s2 in string s1.	strstr(s1, s2);	`s1.find(s2);`

Strings Examples – C++

```cpp
#include <iostream>
#include <string>
using namespace std;
int main() {
  string s1, s2; // empty
  s1 = "Hello";
  s1 += " World!";
  s2 = "Hello World!";

  cout << "String 1 " << s1 << " Length: " <<
  s1.length() << endl;
  cout << "String 2 " << s2 << " Length: " <<
  s2.length() << endl;
  if (s1 == s2)
    cout << "Strings are equal" << endl;
  else
    cout << "Strings are not equal" << endl;
  cout << "First character of s1 is: " << s1[0];
  return 0;
}
```

Output:

```
String 1 Hello World! Length: 13
String 2 Hello World! Length: 13
Strings are equal
First character of s1 is: H
```

Array of Strings Examples – Char Array

```
#include <stdio.h>
int main(void)
{
    char fish[][11] = {"terakihi","snapper","flounder","guppy" };
    printf("%s", fish[1]);
    return 0;
}
```

Output: snapper

Array of Strings Example – String Class

```
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
    string colour[4] = {"Violet", "Red", "Orange", "Yellow"};
    cout << colour[1] << endl;

    return 0;
}
```

Output: Red

Inheritance

Inheritance - Extending Classes

- Just like Java, C++ supports class **inheritance**
- **Sub Class or Derived class** – a class that inherits member fields from another class
- **Super Class or Base Class** – a class whose fields are inherited by sub class
- **The sub class is said to extend the base class**

Inheritance - Extending Classes

- Syntax of extending a single base class:

```
class subclass_name : access_mode baseclass_name {
      class_member_list
};
```

  - *subclass_name* is the identifier given to the sub class being declared
  - *access_mode* controls the access of inherited fields
  - *baseclass_name* is the identifier of the super class being extended

Example

Base Class:

```
class Animal {
public:
  const char *getName() const;
  void sleep();
  void eat(int food);
  Animal();
  Animal(const char *);

protected:
  int age;

private:
  char name[100];
};
```

Sub Class:

```
class Dog : public Animal {
public:
  int bark(int loudness);
  int bite(int strength);
  void run(int speed);
  void eat(int food);
  Dog(const char *n) : Animal(n) {}

private:
  int skills;
};
```

Recap: Member Access Specifier

- **Private members** – can only be accessed by member functions (and friends) and not accessible by descendant classes
- **Public members** – can be accessed outside the class and inherited by descendant classes
- **Protected members** – can only be accessed by member functions (and friends) and inherited by descendant classes
- When member access specifier is not indicated, default access is `private`

Example

```
class Animal {
public:
    …
protected:
    …
private:
    …

};
```

```
class Dog : public Animal {

public:
  int bark(int loudness);
  int bite(int strength);
  void run(int speed);
  void eat(int food);
  Dog(const char *n) : Animal(n) {}

private:
  int skills;
};
```

Access Mode	Base class access specifier		
	public	**protected**	**private**
public	public	protected	Not accessible
protected	protected	protected	
private	private	private	

If the access mode is not used, then it is private by default

Example

```
class Dog : public Animal {
public:
  int bark(int loudness);
  int bite(int strength);
  void run(int speed);


  void eat(int food);

  Dog(const char *n) : Animal(n) {}

private:
  int skills;

};
```

```
class Animal {
public:
  const char *getName() const;
  void sleep();
  void eat(int food);
  Animal();
  Animal(const char *);

protected:
  int age;

private:
  char name[100]; };
```

Member functions specific to sub class Dog

Example

```
class Dog : public Animal {
public:
  int bark(int loudness);
  int bite(int strength);
  void run(int speed);


  void eat(int food);

   Dog(const char *n) : Animal(n) {}

   private:
   int skills;

};
```

```
class Animal {
public:
   const char *getName() const;
   void sleep();
   void eat(int food);
   Animal();
   Animal(const char *);

protected:
   int age;

private:
   char name[100]; };
```

Member function present in base class - will be **overridden** by sub class

Example

```
class Dog : public Animal {
public:
  int bark(int loudness);
  int bite(int strength);
  void run(int speed);

  void eat(int food);

   Dog(const char *n) : Animal(n) {}

private:
   int skills;

};
```

```
class Animal {
public:
   const char *getName() const;
   void sleep();
   void eat(int food);
   Animal();
   Animal(const char *);

protected:
   int age;

private:
   char name[100]; };
```

Member variable specific to sub class Dog

Example

```
class Dog : public Animal {
public:

  int bark(int loudness);
  int bite(int strength);
  void run(int speed);
  void eat(int food);

  Dog(const char *n) : Animal(n) {}

private:
  int skills;
};
```

Constructor of Dog invokes appropriate super class constructor

Unlike Java, C++ does not have super keyword for invoking super class constructor

When an object of derived class is instantiated, first the *base portion* of Derived is constructed (using the Base class constructor). Once the Base portion is finished, the Derived portion is constructed (using the Derived class constructor).

Overriding Member Functions

- Function overriding is a feature that allows us to have a same function in child class which is already present in the parent class

- It is like creating a new version of an old function, in the child class

```
class Dog : public Animal {
public:
  …
  // Overridden function
  void eat(int food){

  }
};
```

```
Dog d;
d.eat(10);
```

Invokes function definition in child class

Prototype same as base class function.
Overridden member function.

Overriding Member Functions

How to invoke base class function:

```
class Dog : public Animal {
public:
  …
  // Overridden function
  void eat(int food);
  …
};
```

```
Animal::eat(10);
```

Within member
Function of derived class

```
Dog d;
d.Animal::eat(10);
```

From an instance

Pointers and Inheritance

```
class Animal {
public:
  void display()
  {
    cout << "display Animal" << endl;
  }

};

class Dog : public Animal {
public:
  void display()
  {
    cout << "display Dog" << endl;
  }
};
```

Valid, Dog *is an* Animal

```
int main (void)
{
    Animal* bptr;
    Dog d;

    bptr = &d;

   // Bound at compile time
    bptr->display();
}
```

Output:
display Animal

Run Time Polymorphism

- Allow a member function to be called **based on the content** of the base class pointer rather than its **type**.
- Implemented using Virtual functions

```
class Animal {
public:
  virtual void display()
   { cout << "display Animal" << endl; }
};
class Dog : public Animal {
public:
  void display()
   { cout << "display Dog" << endl; }
};
```

```
int main (void)
{
    Animal* bptr;
    Dog d;

    bptr = &d;

    // Bound at run time
    bptr->display();
}
```

```
Output:
display Dog
```

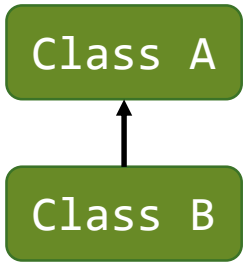
Pure Virtual Function and Abstract Classes

- **Pure virtual function** is declared in the base class without any implementation
- Pure virtual functions must be implemented by a sub class that needs to be instantiated (**concrete)**
- A class that contains at least one **pure virtual function** member is called as an **Abstract class**
- Abstract classes cannot be instantiated

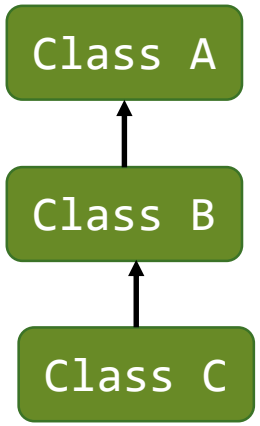
```
class Shape {
public:
  // Pure virtual function
  virtual float draw() = 0;

  // Virtual function
  virtual int getSides() {
    return 1;
  }
};
```

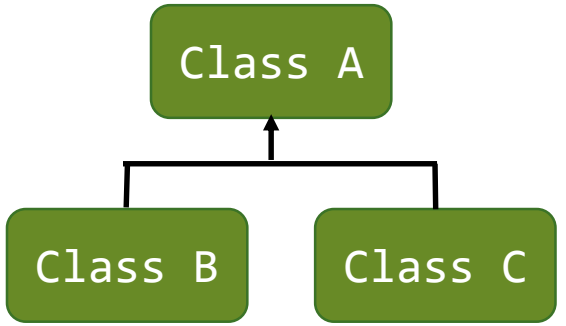
Types of Inheritance



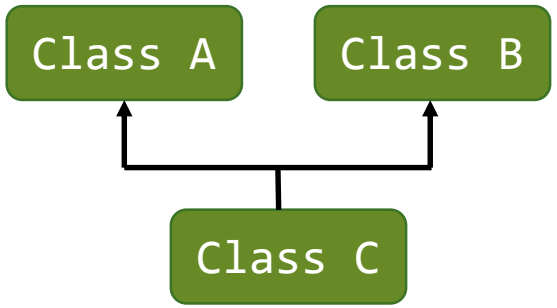
Single



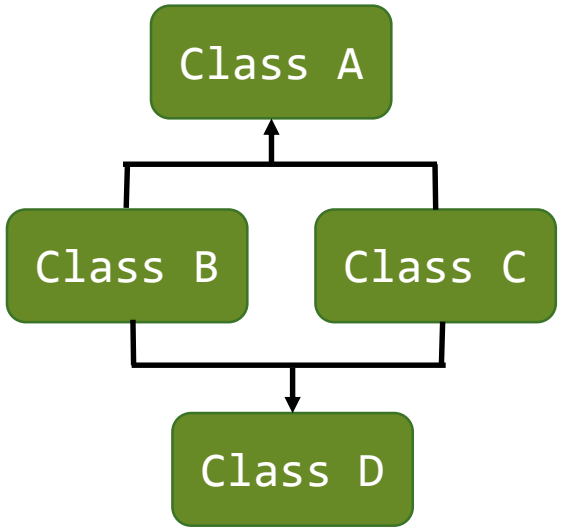
Multi-level



Hierarchical



Multiple



Hybrid

Multiple Inheritance

- C++ supports multiple inheritance

- Syntax for extending multiple classes:

```
class Subclass_name : access_mode1 Baseclass_name1, … ,
  access_modeN Baseclass_nameN {
      class_member_list
};
```

Example

- Suppose that Human and Dog are existing classes

```
class Werewolf : public Human, public Dog {
public:
  void transform();
  …
  Werewolf(const char *n) : Human(n), Dog(n) {}

private:
  int transformCount;
  …
};
```

Member Function Clash

- What happens if base classes of a derived class have a common member function?

```
class A {
public:
  void foo();
  …
};

class B {
public:
  void foo();
  …
};
```

```
class C : public A, public B {
  …
};
```

Class C must override foo(), example:

```
class C : public A, public B {
public:
  void foo() {
    A::foo(); // Use A's implementation of foo!
  }
};
```

Next Lecture

- Containers

- File Handling in C++