12. Polymorphism: Concept, Virtual Functions

Virtual Function

```cpp
class Employee {
    char* name;    char* title;
public:
    Employee (char* nme, char* ttl);
    virtual  void print() { cout << name << " " << ttl; }
};

class Manager : public Employee {
    int level;
    Employee* manages_list;
public:
    Manager(char* nme, char* ttl, int lev);
    virtual void print() {
        Employee::print();
        cout << " " << level; }
};

Employee bruce;
Manager bob;

Employee eptr;
eptr =&bruce;
...
eptr =&bob;
...
eptr-&print(); // What does eptr point to?
    // What gets printed?
```
Binding

Static - Early Binding - Determined at Compile Time
Dynamic - Late Binding - Determined at Run Time

Example:
Employee bruce;
Employee* eptr;
...
bruce.print(); // static
eptr->print(); // dynamic
void SecurityCheck(Employee& emp)
 {emp.print(); // dynamic ...}
void foo(Employee theemp)
 {theemp.print(); // static ...}

Polymorphism: Virtual Functions

Virtual functions Override function definitions in the base class

The derived class may override a virtual function defined in the base class - Should have the same signature and return type

Keep in mind “Substitutability”
Overloading Vs. Overriding

<table>
<thead>
<tr>
<th>Overloaded Functions</th>
<th>Overridden Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same Scope</td>
<td>Different Scope</td>
</tr>
<tr>
<td>Same Name</td>
<td>Same Name</td>
</tr>
<tr>
<td>Different Signature</td>
<td>Same Signature</td>
</tr>
<tr>
<td>Virtual Not Required</td>
<td>Virtual Required</td>
</tr>
</tbody>
</table>

Improper Overriding : Function Hiding

A Derived class function hides (instead of overriding) a base class function with the same name if the derived class function has

- Different Signature
- Same signature & non-virtual in Base
Example of Virtual Functions

class Base {
public: ...
    virtual void f(float x);
    virtual void g(float x);
    void h(float x);
};

class Derived : public Base {
public : ...
    virtual void f(float x);
    virtual void g(int x);
    virtual void h(float x);
};

Derived::f(float);
overrides
Base::f(float);

Derived::g(int);
hides
Base::g(float);

Derived::h(float);
hides
Base::h(float);

Open-Close Principle

“Software Entities (Classes, modules, Functions, etc.)

should be

• Open for Extension (in behavior)

• Closed for Modification (of Code)”
Lab Work: Details provided on-line.