Module 31: Programming in C++

Virtual Function Table

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

Module Objectives

- Understand Virtual Function Table for dynamic binding (polymorphic dispatch)

Module Outline

- Staff Salary Processing: RECAP
  - C Solution using Function Pointers
  - C++ Solution using Polymorphic Hierarchy
  - Comparison of C and C++ Solutions
- Virtual Function Table for Polymorphic Dispatch

Staff Salary Processing: Problem Statement: RECAP (Module 29)

- An organization needs to develop a salary processing application for its staff
- At present it has an engineering division only where Engineers and Managers work. Every Engineer reports to some Manager. Every Manager can also work like an Engineer
- The logic for processing salary for Engineers and Managers are different as they have different salary heads
- In future, it may add Directors to the team. Then every Manager will report to some Director. Every Director could also work like a Manager
- The logic for processing salary for Directors will also be distinct
- Further, in future it may open other divisions, like Sales division, and expand the workforce
- **Make a suitable extensible design**

C Solution: Function Pointers
Engineer + Manager: RECAP (Module 29)

- How to represent Engineers, Managers, and Directors?
  - `struct`
- How to initialize objects?
  - Initialization functions
- How to have a collection of mixed objects?
  - Array of `union`
- How to model variations in salary processing algorithms?
  - `struct`-specific functions
- How to invoke the correct algorithm for a correct employee type?
  - Function switch
  - Function pointers

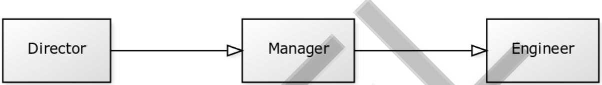
C Solution: Function Pointers
Engineer + Manager + Director

```
#include <stdio.h>
#include <string.h>

typedef enum E_TYPE { Er, Mgr, Dir } E_TYPE;
typedef void (*psFuncPtr)(void *);

typedef struct Engineer { char *name_; } Engineer;
Engineer *InitEngineer(const char *name) { Engineer *e = (Engineer *)malloc(sizeof(Engineer));
    e->name_ = strdup(name); return e;
}
void ProcessSalaryEngineer(void *v) { Engineer *e = (Engineer *)v;
    printf("%s: Process Salary for Engineer\n", e->name_);
}
typedef struct Manager { char *name_; Engineer *reports_[10]; } Manager;
Manager *InitManager(const char *name) { Manager *m = (Manager *)malloc(sizeof(Manager));
    m->name_ = strdup(name); return m;
}
void ProcessSalaryManager(void *v) { Manager *m = (Manager *)v;
    printf("%s: Process Salary for Manager\n", m->name_);
}
typedef struct Director { char *name_; Manager *reports_[10]; } Director;
Director *InitDirector(const char *name) { Director *d = (Director *)malloc(sizeof(Director));
    d->name_ = strdup(name); return d;
}
void ProcessSalaryDirector(void *v) { Director *d = (Director *)v;
    printf("%s: Process Salary for Director\n", d->name_);
}
```

C Solution: Function Pointers
Engineer + Manager + Director

```
typedef struct Staff {
    E_TYPE type_;
    void *p;
} Staff;
int main() {
    psFuncPtr psArray[] = { ProcessSalaryEngineer,
                            ProcessSalaryManager,
                            ProcessSalaryDirector };

    Staff staff[] = { { Er, InitEngineer("Rohit") },
                      { Mgr, InitEngineer("Kamala") },
                      { Mgr, InitEngineer("Rajib") },
                      { Er, InitEngineer("Kavita") },
                      { Er, InitEngineer("Shambhu") },
                      { Dir, InitEngineer("Ranjana") } };

    for (int i = 0; i < sizeof(staff) / sizeof(Staff); ++i)
        psArray[staff[i].type_](staff[i].p);

    return 0;
}
-----
Output:
Rohit: Process Salary for Engineer
Kamala: Process Salary for Manager
Rajib: Process Salary for Manager
Kavita: Process Salary for Engineer
Shambhu: Process Salary for Engineer
Ranjana: Process Salary for Director
```

C++ Solution: Polymorphic Hierarchy: RECAP Engineer + Manager + Director: (Module 30)



- How to represent Engineers, Managers, and Directors?
    - Polymorphic `class` hierarchy
- How to initialize objects?
    - Constructor / Destructor
- How to have a collection of mixed objects?
    - `array` of base class pointers
- How to model variations in salary processing algorithms?
    - Member functions
- How to invoke the correct algorithm for a correct employee type?
    - Virtual Functions

C++ Solution: Polymorphic Hierarchy: RECAP Engineer + Manager + Director: (Module 30)

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

class Engineer {
protected:
    string name_;
public:
    Engineer(const string& name) : name_(name) {}
    virtual void ProcessSalary()
                 { cout << name_ << ": Process Salary for Engineer" << endl; }
};
class Manager : public Engineer {
    Engineer *reports_[10];
public:
    Manager(const string& name) : Engineer(name) {}
    void ProcessSalary()
         { cout << name_ << ": Process Salary for Manager" << endl; }
};
class Director : public Manager {
    Manager *reports_[10];
public:
    Director(const string& name) : Manager(name) {}
    void ProcessSalary()
         { cout << name_ << ": Process Salary for Director" << endl; }
};
```

C++ Solution: Polymorphic Hierarchy: RECAP Engineer + Manager + Director: (Module 30)

```
int main() {
    Engineer e1("Rohit");
    Engineer e2("Kavita");
    Engineer e3("Shambhu");
    Manager m1("Kamala");
    Manager m2("Rajib");
    Director d("Ranjana");

    Engineer *staff[] = { &e1, &m1, &m2, &e2, &e3, &d };

    for (int i = 0; i < sizeof(staff) / sizeof(Engineer*); ++i) staff[i]->ProcessSalary();

    return 0;
}
```

```
Output:
Rohit: Process Salary for Engineer
Kamala: Process Salary for Manager
Rajib: Process Salary for Manager
Kavita: Process Salary for Engineer
Shambhu: Process Salary for Engineer
Ranjana: Process Salary for Director
```

C and C++ Solutions: A Comparison

C Solution	C++ Solution
How to represent Engineers, Managers, and Directors?	How to represent Engineers, Managers, and Directors?
- `structs`	- Polymorphic hierarchy
How to initialize objects?	How to initialize objects?
- Initialization functions	- Ctor / Dtor
How to have a collection of mixed objects?	How to have a collection of mixed objects?
- `array` of `union` wrappers	- `array` of base class pointers
How to model variations in salary processing algorithms?	How to model variations in salary processing algorithms?
- functions for `structs`	- `class` member functions
How to invoke the correct algorithm for a correct employee type?	How to invoke the correct algorithm for a correct employee type?
- Function switch - Function pointers	- Virtual Functions

C and C++ Solutions: A Comparison

C Solution (Function Pointer)

```
#include <stdio.h>
#include <string.h>
typedef enum E_TYPE { Er, Mgr, Dir } E_TYPE;
typedef void (*psFuncPtr)(void *);
typedef struct { E_TYPE type_; void *p; } Staff;

typedef struct { char *name_; } Engineer;
Engineer *InitEngineer(const char *name);
void ProcessSalaryEngineer(void *v);
typedef struct { char *name_; } Manager;
Manager *InitManager(const char *name);
void ProcessSalaryManager(void *v);
typedef struct { char *name_; } Director;
Director *InitDirector(const char *name);
void ProcessSalaryDirector(void *v);
int main() { psFuncPtr psArray[] = {
    ProcessSalaryEngineer,
    ProcessSalaryManager,
    ProcessSalaryDirector };

    Staff staff[] = {
    { Er, InitEngineer("Rohit") },
    { Mgr, InitEngineer("Kamala") },
    { Dir, InitEngineer("Ranjana") } };

    for (int i = 0; i <
        sizeof(staff)/sizeof(Staff); ++i)
        psArray[staff[i].type_](staff[i].p);
    return 0;
}
```

C++ Solution (Virtual Function)

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


class Engineer { protected: string name_;
public: Engineer(const string& name);
    virtual void ProcessSalary(); };
class Manager : public Engineer {
public: Manager(const string& name);
    void ProcessSalary(); };
class Director : public Manager {
public: Director(const string& name);
    void ProcessSalary(); };
int main() {




    Engineer e1("Rohit");
    Manager m1("Kamala");
    Director d("Ranjana");
    Engineer *staff[] = { &e1, &m1, &d };

    for(int i = 0; i <
        sizeof(staff)/sizeof(Engineer*); ++i)
        staff[i]->ProcessSalary();
    return 0;
}
```

Virtual Function Pointer Table

Base Class	Derived Class

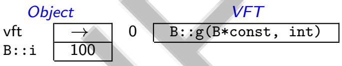
Base Class

```
class B {
  int i;
public:
  B(int i_): i(i_) {}
    void f(int); // B::f(B*const, int)
virtual void g(int); // B::g(B*const, int)
}

B b(100);

B *p = &b;
```

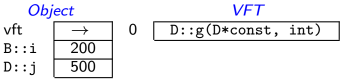
Derived Class

```
class D: public B {
  int j;
public:
  D(int i_, int j_): B(i_), j(j_) {}
    void f(int); // D::f(D*const, int)
    void g(int); // D::g(D*const, int)
}

D d(200, 500);

B *p = &d;
```

b **Object Layout**



d **Object Layout**



Source Expression	Compiled Expression	Source Expression	Compiled Expression
`b.f(15);`	`B::f(&b, 15);`	`d.f(15);`	`D::f(&d, 15);`
`p->f(25);`	`B::f(p, 25);`	`p->f(25);`	`D::f(p, 25);`
`b.g(35);`	`B::g(&b, 35);`	`d.g(35);`	`D::g(&d, 35);`
`p->g(45);`	`p->vft[0](p, 45);`	`p->g(45);`	`p->vft[0](p, 45);`

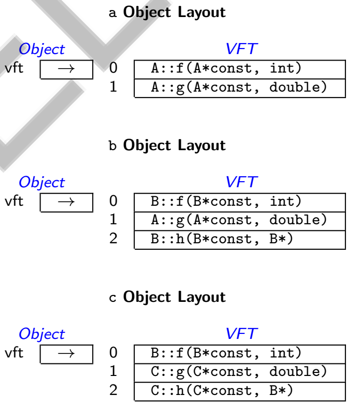
Virtual Function Pointer Table

- Whenever a class defines a virtual function a hidden member variable is added to the class which points to an array of pointers to (virtual) functions called the **Virtual Function Table** (VFT)
- VFT pointers are used at run-time to invoke the appropriate function implementations, because at compile time it may not yet be known if the base function is to be called or a derived one implemented by a class that inherits from the base class
- VFT is class-specific – all instances of the class has the same VFT
- VFT carries the **Run-Time Type Information** (RTTI) of objects

Virtual Function Pointer Table

```
class A { public:
    virtual void f(int) { }
    virtual void g(double) { }
    int h(A *) { }
};
class B: public A { public:
    void f(int) { }
    virtual int h(B *) { }
};
class C: public B { public:
    void g(double) { }
    int h(B *) { }
};

A a; B b; C c;

A *pA; B *pB;
```

Source Expression	**Compiled Expression**
`pA->f(2);`	`pA->vft[0](pA, 2);`
`pA->g(3.2);`	`pA->vft[1](pA, 3.2);`
`pA->h(&a);`	`A::h(pA, &a);`
`pA->h(&b);`	`A::h(pA, &b);`
`pB->f(2);`	`pB->vft[0](pB, 2);`
`pB->g(3.2);`	`pB->vft[1](pB, 3.2);`
`pB->h(&a);`	`pB->vft[2](pB, &a);`
`pB->h(&b);`	`pB->vft[2](pB, &b);`



a **Object Layout**

Object: vft →

	VFT
0	`A::f(A*const, int)`
1	`A::g(A*const, double)`

b **Object Layout**

Object: vft →

	VFT
0	`B::f(B*const, int)`
1	`A::g(A*const, double)`
2	`B::h(B*const, B*)`

c **Object Layout**

Object: vft →

	VFT
0	`B::f(B*const, int)`
1	`C::g(C*const, double)`
2	`C::h(C*const, B*)`

Module Summary

- Leveraging an innovative solution to the Salary Processing Application in C using function pointers, we compare C and C++ solutions to the problem
- The new C solution is used to explain the mechanism for dynamic binding (polymorphic dispatch) based on virtual function tables

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655