



#### Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

# Module 31: Programming in C++

## Virtual Function Table

Instructors: Abir Das and Sourangshu Bhattacharya

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

*{abir, sourangshu}@cse.iitkgp.ac.in*

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**



# Weekly Recap

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

## Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- Understood type casting – implicit as well as explicit – for built-in types, unrelated types, and classes on a hierarchy
- Understood the notions of upcast and downcast
- Understood Static and Dynamic Binding for Polymorphic type
- Understood **virtual** destructors, Pure Virtual Functions, and Abstract Base Class
- Designed the solution for a staff salary processing problem using iterative refinement – starting with a simple C solution and repeatedly refining finally to an easy, efficient, and extensible C++ solution based on flexible polymorphic hierarchy



# Module Objectives

- Introduce a new C solution with function pointers
- Understand Virtual Function Table for dynamic binding (polymorphic dispatch)

Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary



# Module Outline

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- 1 Weekly Recap
- 2 Staff Salary Processing: New C Solution
- 3 Staff Salary Processing: C++ Solution
- 4 C and C++ Solutions: A Comparison
- 5 Virtual Function Pointer Table
- 6 Module Summary



# Staff Salary Processing: New C Solution

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

**Staff Salary  
Processing: New  
C Solution**

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

## Staff Salary Processing: New C Solution



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

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- 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 + Director: RECAP (Module 29)

### Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- How to represent **Engineers**, **Managers**, and **Directors**?
  - Collection of **structs**
- 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

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- In Module 29, we have developed a flat C Solution using *function switch*
- In Module 30, we refined the C Solution to develop two types of C++ Solution using
  - Non-polymorphic hierarchy - employing *function switch*
  - Polymorphic hierarchy - employing *virtual function*
- In Module 29, we had mentioned that in the flat C Solution it is not easy to use function pointers as the processing functions `void ProcessSalaryEngineer(Engineer *)`, `void ProcessSalaryManager(Manager *)`, and `void ProcessSalaryDirector(Director *)` all have different types of arguments and therefore a common function pointer type cannot be defined
- We can work around this by:
  - Passing the staff object as `void *`, instead of `Engineer *`, `Manager *`, or `Director *`
  - Cast it to respective object type in the respective function. That is, cast to `Engineer *` in `ProcessSalaryEngineer(Engineer *)` and so on
  - We can then use a function pointer type `void (*)(void *)`
- We illustrate in the Solution





# C Solution: Function Pointers: Engineer + Manager + Director

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

typedef enum E_TYPE { Er, Mgr, Dir } E_TYPE; // Staff tag type
typedef void (*psFuncPtr)(void *); // Processing func. ptr. type, passing the object by void *
typedef struct Engineer { char *name_; } Engineer; // Engineer Type
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; // Cast explicitly to the staff object
    printf("%s: Process Salary for Engineer\n", e->name_);
}

typedef struct Manager { char *name_; Engineer *reports_[10]; } Manager; // Manager Type
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; // Cast explicitly to the staff object
    printf("%s: Process Salary for Manager\n", m->name_);
}

typedef struct Director { char *name_; Manager *reports_[10]; } Director; // Director Type
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; // Cast explicitly to the staff object
    printf("%s: Process Salary for Director\n", d->name_);
}

}

$S20202: Software Engineering
```



# C Solution: Function Pointers: Engineer + Manager + Director

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

```
typedef struct Staff {
    E_TYPE type_; // Staff tag type
    void *p;      // Pointer to staff object
} Staff;        // Staff object wrapper

int main() {
    // Array of function pointers
    psFuncPtr psArray[] = { ProcessSalaryEngineer, ProcessSalaryManager, ProcessSalaryDirector };

    // Array of staffs
    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_] // Pick the right processing function for the tag - staff type
                                (staff[i].p); // Pass the pointer to the object - implicitly cast to void*
}
```

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: Advantages and Disadvantages: RECAP (Module 26)

## Annotated for Function Pointers

Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- **Advantages**
  - Solution exists!
  - Code is well structured – has patterns
- **Disadvantages**
  - Employee data has scope for better organization
    - ▷ No encapsulation for data
    - ▷ Duplication of fields across types of employees – possible to mix up types for them (say, `char *` and `string`)
    - ▷ Employee objects are created and initialized dynamically through `Init...` functions. How to release the memory?
  - Types of objects are managed explicitly by `E_Type`:
    - ▷ Difficult to extend the design – addition of a new type needs to:
      - Add new type code to `enum E_Type`
      - Add a new pointer field in `struct Staff` for the new type
      - Add a new case (`if-else` or `case`) based on the new type: **Removed using function pointer**
    - ▷ Error prone – developer has to decide to call the right processing function for every type (`ProcessSalaryManager` for `Mgr` etc.): **Removed using function pointer**
  - Unable to use Function Pointers as each processing function takes a parameter of different type - no common signature for dispatch
- **Recommendation**
  - Use classes for encapsulation on a hierarchy



# Staff Salary Processing: C++ Solution

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

**Staff Salary  
Processing: C++  
Solution**

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

## Staff Salary Processing: C++ Solution



# 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)

### Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

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

class Engineer {
protected:
    string name_;
public:
    Engineer(const string& name) : name_(name) { }
    virtual ~Engineer() { }
    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)

### Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

```
int main() {  
    Engineer e1("Rohit"), e2("Kavita"), e3("Shambhu");  
    Manager m1("Kamala"), 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();  
}
```

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

Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

## C and C++ Solutions: A Comparison





# C and C++ Solutions: A Comparison

Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

## C Solution

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

## C++ Solution

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



# C and C++ Solutions: A Comparison

Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

## C Solution (Function Pointer)

```
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, // Function
    ProcessSalaryManager, // pointer
    ProcessSalaryDirector }; // array
    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);
}
```

## C++ Solution (Virtual Function)

```
class Engineer { protected: string name_;
public: Engineer(const string& name);
        virtual void ProcessSalary(); };
        virtual ~Engineer(); };
class Manager : public Engineer {
public: Manager(const string& name);
        void ProcessSalary(); };
class Director : public Manager {
public: Director(const string& name);
        void ProcessSalary(); };
int main() {
    // Function pointer array is subsumed in
    // virtual function tables of classes

    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();
}
```



# Virtual Function Pointer Table

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

## Virtual Function Pointer Table



# How do virtual functions work?

- The C Solution with function pointers gives us the lead to implement virtual functions. Here
  - We have used an array of function pointers (`psFuncPtr psArray[]`) to keep the processing functions (`void ProcessSalaryEngineer(Engineer *)`, `void ProcessSalaryManager(Manager *)`, and `void ProcessSalaryDirector(Director *)`) indexed by the type tag (`enum E_TYPE { Er, Mgr, Dir }`)
  - In C++, every class is a separate type - so the tag can be removed if we bind this table (**Virtual Function Table** or **VFT**) with the class
  - Every class can have a VFT with its appropriate processing function pointer put there
  - By override, all these functions can have the same signature (`void ProcessSalary()`) and can be called through the same expression (`(Engineer *)->ProcessSalary()`)
- We now illustrate Virtual Function Table through simple examples to show how does it work for inherited, overridden and overloaded member functions



# Virtual Function Pointer Table

Module 31

Instructors: Abir Das and Sourangshu Bhattacharya

Weekly Recap

Objectives & Outline

Staff Salary Processing: New C Solution

Staff Salary Processing: C++ Solution

C and C++ Solutions: A Comparison

Virtual Function Pointer Table

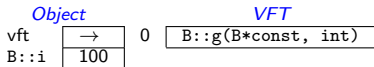
Module Summary

## 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;
```

### b Object Layout



### Source Expression

```
b.f(15);
p->f(25);
b.g(35);
p->g(45);
```

### Compiled Expression

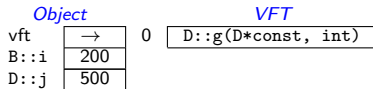
```
B::f(&b, 15);
B::f(p, 25);
B::g(&b, 35);
p->vft[0](p, 45);
```

## 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;
```

### d Object Layout



### Source Expression

```
d.f(15);
p->f(25);
d.g(35);
p->g(45);
```

### Compiled Expression

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



# Virtual Function Pointer Table

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

- 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

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

Module Summary

```
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

```
pA->f(2);
pA->g(3.2);
pA->h(&a);
pA->h(&b);
```

```
pB->f(2);
pB->g(3.2);
pB->h(&a);
pB->h(&b);
```

### Compiled Expression

```
pA->vft[0] (pA, 2);
pA->vft[1] (pA, 3.2);
A::h(pA, &a);
A::h(pA, &b);
```

```
pB->vft[0] (pB, 2);
pB->vft[1] (pB, 3.2);
pB->vft[2] (pB, &a);
pB->vft[2] (pB, &b);
```

## a Object Layout

Object

vft →

VFT

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

## b Object Layout

Object

vft →

VFT

0	B::f(B*const, int)	Overridden
1	A::g(A*const, double)	Inherited
2	B::h(B*const, B*)	Overloaded

## c Object Layout

Object

vft →

VFT

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



# Module Summary

## Module 31

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Weekly Recap

Objectives &  
Outline

Staff Salary  
Processing: New  
C Solution

Staff Salary  
Processing: C++  
Solution

C and C++  
Solutions: A  
Comparison

Virtual Function  
Pointer Table

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 with function pointers is used to explain the mechanism for dynamic binding (polymorphic dispatch) based on **virtual** function tables