ECE326 – TUTORIAL 6

PREPARED BY MARTIYA ZARE JAHROMI

AGENDA

- Exercise 4
- Exercise 5

EXERCISE 4 – 1. TRUE OR FALSE

1. With C3 Linearization, Python completely solves the diamond problem. T F

EXERCISE 4 – 1. TRUE OR FALSE

2. 0x8888FEDC is a 4-byte aligned address. **T** **F**

$(C)_{16} = (12)_{10}$

EXERCISE 4 – 1. TRUE OR FALSE

3. Suppose class A is inherited by class B, and C, monotonicity guarantees that A will behave the same for both B and C.

T F

EXERCISE 4 – 1. TRUE OR FALSE

4. Adding a new pure virtual function to a base class with many existing derived classes is an example of a fragile base class problem. **T** **F**

EXERCISE 4 – 1. TRUE OR FALSE

5. The main difference between delegation and type embedding is that with type embedding, you can no longer reference the embedded member by name. T F

EXERCISE 4 – 2. TRUE OR FALSE

1. Which of the following are true about mixins?

a)It requires subclass to complete its implementation. Not Necessarily.

b)It can contain both member variables and functions.

c)It is used as a super type to the derived class.

d)Using it requires method forwarding. Only composition requires forwarding.

e)The order in which mixins are composed may change behaviour of the subclass.

EXERCISE 4 – 2. TRUE OR FALSE

2. Java only supports single inheritance with runtime polymorphism. Which of the following is true?

a) Java does not support mixins. Mixin requires multiple inheritance.

b) Java does not need virtual tables. Still required to implement dynamic dispatch.

c) Casting pointers (internally, Java does not expose pointers to programmers) in Java will never require point offsetting. This is a requirement for multiple inheritance.

d) Java does not need to deal with inheritance-related ambiguity.

e) Java does not have method resolution order. Only late binding languages require MRO.

EXERCISE 4 – 3. VIRTUAL BASE CLASS IN C++

3. Draw the data layout of class X (include padding assuming 8-byte alignment, and write down the size of each sub-structure) and all the virtual tables generated for class X and its ancestors.

EXERCISE 4 – 3. VIRTUAL BASE CLASS IN C++

- ```
  struct B {
    int b1;
    int b2;
    virtual void foo() { cout << "A.foo"; }
    virtual ~A() {}
  };

  struct P : virtual public B {
    long p1;
    virtual void foo() override { cout << "P.foo"; }
  };

  struct Q : public P {
    int q1;
  };

  struct N : virtual public B {
    char n1[30];
  };

  struct X : public N, public Q {
    int x1;
    virtual void foo() override { cout << "X.foo"; }
  };
  ```
  ```

EXERCISE 4 – 3. VIRTUAL BASE CLASS IN C++

`struct X`

B::b2 : int B::b1 : int B::__vptr	16 bytes
X::x1 : int	4 bytes
Q::q1 : int	4 bytes
P::p1 : long P::__vptr	16 bytes
padding: 2 bytes N::n1 : char[30] N::__vptr	40 bytes

`X::B::vtable`

virtual base offset: 0
offset to “bottom”: 64
typeinfo = typeinfo(X)
X::foo
X::~X()

`X::N::vtable`

virtual base offset: 64
offset to “bottom”: 0
typeinfo = typeinfo(X)
X::foo
X::~X()

`X::P::vtable`

virtual base offset: 24
offset to “bottom”: 40
typeinfo = typeinfo(X)
X::foo
X::~X()

EXERCISE 4 – 3. METHOD RESOLUTION ORDER

L[A] = (A, o)

L[B] = (B, o)

L[C] = (C, o)

L[D] = (D, o)

L[E] = (E, o)

L[P] = (P, A, B, C, o)

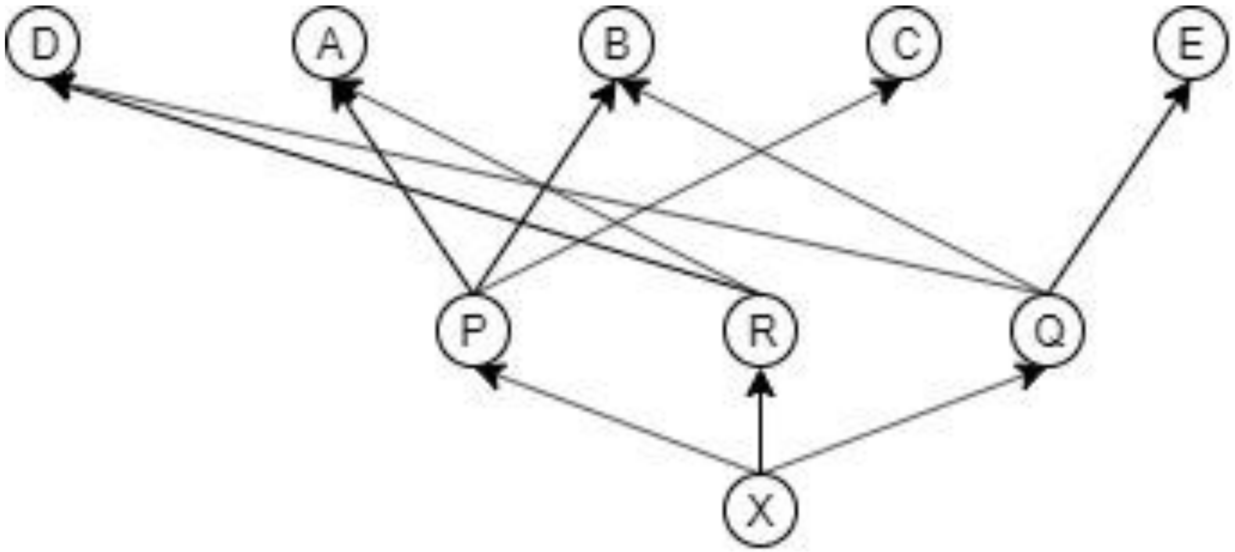
L[Q] = (Q, D, B, E, o)

L[R] = (R, D, A, o)

L[X]

= (X, merge((P, A, B, C, o), (R, D, A, o), (Q, D, B, E, o)))

= (X, P, R, Q, D, A, B, C, E, o)



EXERCISE 5 – 1. TRUE OR FALSE

1. Generic programming is a subset of metaprogramming.

   T **F**

Neither is a subset of each other, but they do have overlaps.

EXERCISE 5 – 1. TRUE OR FALSE

2. If no deep copy is required (e.g. class has no pointer), move semantics performs no better than copy semantics.

T F

EXERCISE 5 – 1. TRUE OR FALSE

3. If template specialization is not used (i.e. not instantiated), its code is not generated for the final executable. **T** **F**

EXERCISE 5 – 1. TRUE OR FALSE

4. For template T foo(), you can write int a = foo() to instantiate the function template foo with an int parameter . **T** **F**

Have to write foo<int>() because C++ does not do type inference based on return type.

EXERCISE 5 – 1. TRUE OR FALSE

5. The new operator in C++ couples heap allocation and constructor invocation. T F

EXERCISE 5 – 2. SHORT ANSWERS

1. Use container_of to return a pointer to the parent object of member field base.

```
struct base {
  int x, y, z;
};

struct derived {
   int a;
   struct base b;
   char c[10];
};

struct derived * get_derived(struct base * ptr) {
   return container_of(ptr, struct derived, b);
  }
```

EXERCISE 5 – 2. SHORT ANSWERS

2. Implement binary search algorithm using a function template, assume the array is sorted

and return -1 upon not found.

```
template<typename T>  /* find index of val in array of size n */
int binary_search(const T & val, T * array, int n) {
        int top = n-1;
        int bot = 0;
        while (bot <= top) {
                int mid = (top + bot)/2;
                if (array[mid] == val)
                return mid;
                else if (array[mid] < val)
                        bot = mid+1;
                else
                        top = mid-1;
        }

        return -1;
}
```

EXERCISE 5 – 2. SHORT ANSWERS

3. Implement a template class named Triple that is a tuple of 3 elements of the same type. Overload enough operators so that binary search template you implemented above can be instantiated for Triple. Use lexicographical order.

```
template<typename T>
struct Triple {
        T a, b, c;
        Triple() : a(0), b(0), c(0) {}
        Triple(T && a, T && b, T && c)
          : a(std::move(a))
          , b(std::move(b))
          , c(std::move(c))
        {}
        bool operator==(const Triple<T> & rhs) {
                return a == rhs.a && b == rhs.b
&& c == rhs.c;
        }
        bool operator<(const Triple<T> & rhs) {
                if ( a < rhs.a )
                        return true;
                else if ( a > rhs.a )
                        return false;
                else if ( b < rhs.b )
                        return true;
                else if ( b > rhs.b )
                        return false;
                else if ( c < rhs.c )
                        return true;
                /* c >= rhs.c */
                return false;
        }
};
```

EXERCISE 5 – 3. GENERIC PROGRAMMING

Create a generic Queue class without using templates. Implement the Queue using a singly linked list, with the member functions, push_back, that pushes new elements to end of the queue, front, which returns the first element of the queue, and pop_front, which removes the first element of the queue.

EXERCISE 5 – 3. GENERIC PROGRAMMING

```
class Queue {
        struct Node {
                Node * next;
                void * data;

                Node(void * data, Node * next=nullptr)
                        : next(next) , data(data) {}
                ~Node() { /* managed by Queue */ }
        } * head, * tail;
        void (* dest_f)(void *);

public:
        Queue(void (* destroy)(void *))
                : head(nullptr)
                , tail(nullptr)
                , dest_f(destroy)
        {}
```

```
~Queue() {
        Node * curr = head;
        while (curr != nullptr) {
                Node * temp = curr;
                curr = curr->next;
                dest_f(temp->data);
                delete temp;
        }
}

void * front() {
        if (head == nullptr) {
                return nullptr;
        }
        return head->data;
}
```

EXERCISE 5 – 3. GENERIC PROGRAMMING

```
        bool pop_front() {
                Node * node;

                if (head == nullptr) {
                        return false;
                }

                node = head;
                head = head->next;
                if (head == nullptr) {
                        tail = nullptr;
                }

                delete node;
                return true;
        }
};
```

EXERCISE 5 – 4. TEMPLATE PROGRAMMING

Using the generic Queue made in Question 3, write a FIFO class template, which allows type-safe use of the generic Queue class for any parameterized type. Use move semantics for push_back instead of copy semantics.

EXERCISE 5 – 4. TEMPLATE PROGRAMMING

```
template<typename T>
class Fifo : private Queue {
        static void destroy(void * ptr) {
                delete (T *)ptr;
        }

public:
        Fifo() : Queue(&Fifo<T>::destroy) {}

        bool push_back(T && elem) {
                return Queue::push_back(new T(std::move(elem)));
        }

        T * front() {
                return (T *)Queue::front();
        }

        bool pop_front() {
                return Queue::pop_front();
        }
};
```

Questions?