Question 1:

Write the definition for macro offsetof (type, member)

Solution:

#define myoffsetof(type, member) (size\_t) (((type\*)(0))->member)

1. (type\*)(0)

Cast integer 0 to a pointer to type

1. (type\*)(0)->member

Deference the pointer to a certain member of the type.

1. &(((type\*)(0))->member)

Take the address

1. (size\_t) &(((type\*)(0))->member)

Cast the address to appropriate data type

Reference: <http://www.embedded.com/design/prototyping-and-development/4024941/Learn-a-new-trick-with-the-offsetof--macro>

Question 2:

What is the output?

class A

{

A() { std::cout << “creating A” << std::endl; }

~A() { std::cout << “deleting A” << std::endl; }

};

class B : public A

{

B() { std::cout << “creating B” << std::endl; }

~B() { std::cout << “deleting B” << std::endl; }

};

int main ()

{

{

B deriObj;

}

A\* pA = new B();

delete pA;

return 0;

}

Solution:

creating A

creating B

deleting B

deleting A

creating A

creating B

deleting A 🡪 no virtual destructor, only delete the base part

Question 3:

In 32-bit system, sizeof(A) = ?

In 64-bit system, sizeof(A) = ?

struct A

{

char a;

char b[2];

long c;

char\* p;

};

Solution:

In 32-bit, sizeof(A) = 1 + 1(padding) + 2 + 4 + 4 = 12

In 64-bit, sizeof(A) = 1 + 1(padding) + 2 + 4 + 8 = 16

Reference :

<http://www.geeksforgeeks.org/structure-member-alignment-padding-and-data-packing/>

Question 4:

Assume you are getting a stream of quotes with the information below. Design a program to maintain an orderbook which has 4 levels on each side.

struct Quote

{

uint32\_t timestamp;

uint32\_t orderid;

char           type;     // New, Modify, Delete

char           side;       // Buy, Sell

uint32\_t       price;

uint32\_t       size;

};

My Solution:

Data Structures I use for the problem are,

1. **struct Quote :**

Given by the problem.

1. **Struct OrderInfo :**

Stores the information for an existing order, including **orderId**, **price**, **size** (remained size to get filled).

1. **struct LevelInfo :**

Stores **price** and **size** at a certain level of the book. It also contains **std::deque<OrderInfo>** that maintains the waiting orders at this level.

1. **std::deque<LevelInfo> :**

There are two containers, **bidLevels** and **askLevels**, contains the most current information of the book on each side. The best price is always at the front. When market moves, we need to add new levelInfoto the front of one side and remove the front levelInfo on the other side. It makes deque a good candidate for these operations.

1. **std::unordered\_map<orderId, OrderInfo>:**

Maintains all existing orders.

Assumptions:

1. The modify order

We only change price and size, we don’t change the side of the orders.

1. The market order

A buy/sell order can only be placed on best offer(ask[0]) / bid(bid[0]).

**Logic of the program**

if (type == “New”)

{

If (buy order)

{

// Add the order into order map

If (market order)

{

If (order size > best offer size) // market moves up!

{

// clean up the best offer level

// create new best bid level for the remain orders

// update order map

}

Else

{

// part of best orders get filled, update waiting queue

// update order map

}

}

Else

{

// search bidLevel using price

If (found the level)

{

// update level’s size

// update level’s waiting queue

}

Else

{

// creating new level

}

}

}

if (Sell order)

{

// Same as above

}

}

if (type == “Modify”)

{

If (Buy order)

{

// search the existing bidLevel using the price

// search the existing order from waiting queue when price is found

If (modify the price or modify both)

{

// remove the order from the queue

// update level’s size

// placing a new order on other price level

}

If (only modify the size)

{

// update level’s size

// update level’s waiting queue

}

}

If (Sell order)

{

Same as above

}

}

if (type == “Delete”)

{

If (Buy order)

{

// search the existing bidLevel using the price

// search the existing order from waiting queue when price is found

// remove the order from the queue

// update level’s size

// remove order from order map

}

If (Sell order)

{

Same as above

}

}