Name: Tian Zheng

Student number: 155 394 174

Email: tzheng26@myseneca.ca

(1.) The difference between the implementations of a composition and an aggregation.

In composition, the composer creates the component object(s) and/or allocates memory for it in its constructor, and deallocates the memory in its destructor.

For example, in constructor:

MessagePack::MessagePack(Message\*\* msgsPtr, size\_t size) {

if (msgsPtr != nullptr && size > 0) {

size\_t empty;

messages = new Message[size];

for (size\_t i = 0, j = 0;j < size;i++, j++) {

if (msgsPtr[j]->empty())

j++;

messages[i] = \*msgsPtr[j];

empty = j - i;

}

msgsSize = size - empty;

}

else {

\*this = MessagePack();

}

}

messages = new Message[size]; -> Dynamically allocate memory for the array of Message objects

for (size\_t i = 0, j = 0;j < size;i++, j++) {…} -> Create Message objects, each of which is an element in the array

And an example in the destructor:

//Deallocate the memory for the array of Message objects

MessagePack::~MessagePack() {

delete[] messages;

messages = nullptr;

}

While in aggregation, the aggregator does not have control over the aggregatee. For example, in aggregator’s constructor:

Notifications::Notifications(const int srcMax) {

if (srcMax > 0) {

maxAdd = srcMax;

msgsptr = new const Message\*[srcMax];

for (int i = 0;i < srcMax;i++)

msgsptr[i] = nullptr;

accurateAdd = 0;

}

else { \*this = Notifications(); }

}

msgsptr = new const Message\*[srcMax]; -> Dynamically allocation of memory for an array of pointers to Message objects, not an array of Message objects

for (int i = 0;i < srcMax;i++){…} -> Create pointers, not Message objects

(2.) The difference between the implementations of move and copy functions in a composition and an aggregation.

In a composition, the target of move and copy functions are the component. For example:

MessagePack& MessagePack::operator= (const MessagePack& src) {

if (this != &src) {

delete[] messages;

messages = new Message[src.msgsSize];

for (size\_t i = 0; i < src.msgsSize;i++) {

messages[i] = src.messages[i];

}

msgsSize = src.msgsSize;

}

return \*this;

}

messages[i] = src.messages[i]; -> Message objects are being copied.

MessagePack& MessagePack::operator=(MessagePack&& src) {

if (this != &src) {

delete[] messages;

messages = src.messages;

msgsSize = src.msgsSize;

src.messages = nullptr;

src.msgsSize = 0;

}

return \*this;

}

messages = src.messages; -> The address to an array of Message objects are being copied.

While in aggregation, the target of the copy and move functions is not the aggregatee object. For example:

Notifications& Notifications::operator=(const Notifications& src) {

if (this != &src) {

delete[]msgsptr;

msgsptr = new const Message\*[src.maxAdd];

for (int i = 0;i < src.accurateAdd;i++)

msgsptr[i] = src.msgsptr[i];

maxAdd = src.maxAdd;

accurateAdd = src.accurateAdd;

}

return \*this;

}

msgsptr[i] = src.msgsptr[i]; -> pointers are being copied, not the Message object itself

Notifications& Notifications::operator=(Notifications&& src) {

if (this != &src) {

delete[] msgsptr;

msgsptr = src.msgsptr;

maxAdd = src.maxAdd;

accurateAdd = src.accurateAdd;

src.msgsptr = nullptr;

src.maxAdd = 0;

src.accurateAdd = 0;

}

return \*this;

}

msgsptr = src.msgsptr; -> The pointer to an array of pointers is being copied, not the pointer to an array of Message objects.

3. Except the above two differences, I have also learned that after dynamically allocating memory of an array of pointers, each element in the array, i.e. each pointer, needs to be initialized to nullptr. It prevents the pointers in the array from pointing to random address and helps to keep memory safe.

Correction for week 5 quiz:

Question 1:

Error handling identifies exceptions to normal executions and handles them before a crash.  Complete the following code to handle a divide by zero error and display the message "divide by zero error" to the user:

int divide(int x, int y) {

if (y == 0)

        \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

else

        return x / y;

}

int main () {

int dividend, divisor, quotient = 0;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

quotient = divide(dividend, divisor);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

std::cout << "Quotient = " << quotient << std::endl;

}

Correction:

throw “divide by zero error”;

try{

}catch(const char\* msg) { std::cout<<msg<<std::endl;}

Question 2:

Write a Lambda express that uses "capture-by-value" to add a function parameter 'i' to an external captured 'x' variable.

Correction:

[=](i){return i+k; }