**Lameck Onyango Oyare**

**MSc. Software Engineering**

**Reg.SCT313 – 3506/2017**

1. Write a C++ code stub and simulate the management of the memory portion allocated to a running program and explain how such space is allocated and bounded.

int \*ptr = new int; // dynamically allocate an integer and assign the address to ptr so we can access it later

int \*ptr = new int; // dynamically allocate an integer and assign the address to ptr so we can access it later

1. Define what a stack overflow is and show how it can be mitigated.

The stack has a limited size, and consequently can only hold a limited amount of information. On Windows, the default stack size is 1MB. On some unix machines, it can be as large as 8MB. If the program tries to put too much information on the stack, stack overflow will result. **Stack overflow** happens when all the memory in the stack has been allocated -- in that case, further allocations begin overflowing into other sections of memory.

**how it can be mitigated**

Be very carefull with recursion,

Do not create huge arrays on stack, this might be hidden arrays like a very big array as a class field. Its always better to use vector.

Make sure your stack size is optimal, this is more important in embeded platforms. If you thread does not do much, then give it small stack, otherwise use larger. I know reservation should only take some address range - not physical memory.

C). Describe what a memory leak is and how can it occurs. Write a C++ program stub to illustrate this phenomenon and show how it can be mitigated

A memory leak occurs when a piece (or pieces) of memory that was previously allocated by a programmer is not properly deallocated by the programmer. Even though that memory is no longer in use by the program, it is still “reserved”, and that piece of memory can not be used by the program until it is properly deallocated by the programmer.

|  |
| --- |
| /\* Function with memory leak \*/  #include <stdlib.h>    void f()  {     int \*ptr = (int \*) malloc(sizeof(int));       /\* Do some work \*/       return; /\* Return without freeing ptr\*/  } |

Run on IDE

To avoid memory leaks, memory allocated on heap should always be freed when no longer needed.

|  |
| --- |
| /\* Function without memory leak \*/  #include <stdlib.h>;    void f()  {     int \*ptr = (int \*) malloc(sizeof(int));       /\* Do some work \*/       free(ptr);     return;  } |

1. Abstract Classes are commonly used in C++. Why are they used and what is the C++ syntax for their use?

Abstract Class is a class, which contains atleast one Pure Virtual function in it. Abstract classes are used to provide an Interface for its sub classes. Classes inheriting an Abstract Class must provide definition to the pure virtual function; otherwise they will also become abstract class.

// Base class

class Shape {

public:

// pure virtual function providing interface framework.

virtual int getArea() = 0;

void setWidth(int w) {

width = w;

}

void setHeight(int h) {

height = h;

}

protected:

int width;

int height;

};

// Derived classes

class Rectangle: public Shape {

public:

int getArea() {

return (width \* height);

}

};

class Triangle: public Shape {

public:

int getArea() {

return (width \* height)/2;

}

};

int main(void) {

Rectangle Rect;

Triangle Tri;

Rect.setWidth(5);

Rect.setHeight(7);

// Print the area of the object.

cout << "Total Rectangle area: " << Rect.getArea() << endl;

Tri.setWidth(5);

Tri.setHeight(7);

// Print the area of the object.

cout << "Total Triangle area: " << Tri.getArea() << endl;

return 0;

}

1. Write a section of C++ code that demonstrates the use of dynamic binding. What does object serialization mean?

refers to linking a procedure call to the code that will be executed only at run time. The code associated with the procedure in not known until the program is executed, which is also known as late binding.

#include <iostream.h>  
int Square(int x)  
{ return x\*x; }  
int Cube(int x)  
{ return x\*x\*x; }  
int main()  
{  
int x =10;  
int choice;  
do  
{  
cout << "Enter 0 for square value, 1 for cube value: ";  
cin >> choice;  
} while (choice < 0 || choice > 1);  
int (\*ptr) (int); switch (choice)  
{  
case 0: ptr = Square; break;  
case 1: ptr = Cube; break;  
} cout << "The result is: " << ptr(x) << endl;  
return 0; }

Object Serilization means converting the data into binary data so as to be able to transfer it over the network and then retrieve it back to its original form on the other end by deserialization.

1. Using a clear example distinguish between a protected and private variable and outline one key benefit of such implementation

* Public variables, are variables that are visible to all classes.
* Private variables, are variables that are visible only to the class to which they belong.
* Protected variables, are variables that are visible only to the class to which they belong, and any subclasses.
* "private" means that only methods of this class can access the member. "protected" means that methods of this class, or of derived classes, can access the member.

We have declared our string and integer to be public. This means that any object in the system can change the balance (setting it to zero, or even giving us a negative balance). This could cause the program to fall over, even though we wrote code in our constructor to prevent negative balances.

Instead, we should have provided a getBalance/setBalance method, and made our balance private or proteced. Other objects can still access the data, but they can't put invalid data in.

public class bank\_balance

{

public String owner;

private int balance;

public bank\_balance( String name, int dollars )

{

owner = name;

if (dollars >= 0)

balance = dollars;

else

dollars =0;

}

public int getBalance()

{

return balance;

}

public void setBalance(int dollars)

{

if (dollars >= 0)

balance = dollars;

else

dollars = 0;

}

}

1. Describe three benefits of OO programming over other programming paradigms

* ***Code Reuse and Recycling***: Objects created for Object Oriented Programs can easily be reused in other programs.
* ***Encapsulation (part 1)***: Once an Object is created, knowledge of its implementation is not necessary for its use. In older programs, coders needed understand the details of a piece of code before using it (in this or another program).
* ***Encapsulation (part 2)***: Objects have the ability to hide certain parts of themselves from programmers. This prevents programmers from tampering with values they shouldn't. Additionally, the object controls how one interacts with it, preventing other kinds of errors. For example, a programmer (or another program) cannot set the width of a window to -400.
* ***Design Benefits***: Large programs are very difficult to write. Object Oriented Programs force designers to go through an extensive planning phase, which makes for better designs with less flaws. In addition, once a program reaches a certain size, Object Oriented Programs are actually *easier* to program than non-Object Oriented ones.
* ***Software Maintenance:*** Programs are not disposable. Legacy code must be dealt with on a daily basis, either to be improved upon (for a new version of an exist piece of software) or made to work with newer computers and software. An Object Oriented Program is much easier to modify and maintain than a non-Object Oriented Program. So although a lot of work is spent before the program is written, less work is needed to maintain it over time.

2. Improved software maintainability: For the reasons mentioned above, object oriented software is also easier to maintain. Since the design is modular, part of the system can be updated in case of issues without a need to make large-scale changes.

3. Faster development: Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed during projects is also reusable in future projects.

4. Lower cost of development: The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of development.

5. Higher-quality software: Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Although quality is dependent upon the experience of the teams, objectoriented programming tends to result in higher-quality software.

1. Create a Carpet class that contains data fields that store a Carpet’s length, width, and price and count to keep track of all carpets on sale. The Carpet class will contain public functions that set the length and width of the carpet and the price is determined by the carpets area. Provide a setPrice() function that sets e value of price any time the length or width changes and that which is only invoked by the class and not a client function. Should the setPrice function and count be be private or public. Explain.

When you create a class, usually you want to make data items private, to control how they are used, and to make functions public to provide a means to access and manipulate the data. However, if you have a reason to do so, you are free to make particular data items public. Similarly, not all functions are public.

When you think of real-world objects, such as kitchen appliances, there are many functions you control through a public interface: adjusting the temperature on a refrigerator or oven, setting a cycle on a dishwasher, and so on. However, there are other functions that appliances encapsulate: a freezer might defrost itself without your help, and a dishwasher automatically switches from the wash to the rinse cycle. With objects you create, functions also can be private if you choose.

#include<iostream>  
using namespace std;  
// declaration section

class Carpet  
{  
 private:

int length;  
int width;  
double price;  
void setPrice();

public:

int getLength();  
int getWidth();  
double getPrice();  
void setLength(int);  
void setWidth(int);

};

    // implementation section  
int Carpet::getLength()  
{  
return length;  
}  
int Carpet::getWidth()  
{  
return width;  
}  
double Carpet::getPrice()  
{  
return price;  
}  
void Carpet::setLength(int len)  
{  
length = len;  
setPrice();  
}  
void Carpet::setWidth(int wid)  
{  
width = wid;  
setPrice();  
}  
void Carpet::setPrice()  
{  
const int SMALL = 12;  
const int MED = 24;  
const double PRICE1 = 29.99;  
const double PRICE2 = 59.99;  
const double PRICE3 = 89.99;  
int area = length \* width;  
if(area <= SMALL)  
price = PRICE1;  
else  
if(area <= MED)  
price = PRICE2;  
else  
price = PRICE3;  
}

int main()  
{  
Carpet aRug;  
const char QUIT = 'Q';  
char dim;  
int length;  
int width;  
aRug.setLength(1);  
aRug.setWidth(1);  
cout << "Enter L to enter length or " <<  
"W to enter width or " <<  
QUIT << " to quit > ";  
cin >> dim;  
while(dim != QUIT)  
{  
if(dim == 'L')  
{  
cout << "Enter a length > ";  
cin >> length;  
aRug.setLength(length);  
cout << "Length is " << aRug.getLength() <<  
" Width is " << aRug.getWidth() << endl <<  
"Price is " << aRug.getPrice() << endl;  
}  
else  
{  
cout << "Enter a width > ";  
cin >> width;  
aRug.setWidth(width);  
cout << "Length is " << aRug.getLength() <<  
" Width is " << aRug.getWidth() << endl <<  
"Price is " << aRug.getPrice() << endl;  
}  
cout << "Enter L to enter length or W " <<  
"to enter width or " <<  
QUIT << " to quit > ";  
cin >> dim;  
}  
return 0;  
}

1. Define a class named Customer that holds private fields for a customer ID number, last name, first name, and credit limit. Include four public functions that each set one of the four fields. Do not allow any credit limit over $10,000. Include a public function that displays a Customer’s data. Write a main () function in which you declare a Customer, set the Customer’s fields, and display the results Write a main()function that declares an array of five Customer objects. Prompt the user for values for each Customer, and display all five Customer objects.

**Program Plan**

* Class: Customer
* Member functions
* Data members

Declare: Int customerId

Declare: char lastName

Declare: char firstName

Declare: double creditLimit

* Member functions

Void displayCustomerData();

Void setCustomerId(int num);

Void setLastName(char name[]);

Void setCreditLimit(double CL);

If (CL < 10,000) creditLimit=CL

#include<iostream.h>  
#include<conio.h>  
#include<string.h>  
class Customer  
{  
private:  
   int customerIdNum;  
   char lastName[15];  
   char firstName[15];  
   double creditLimit;  
public:  
   void displayCustomerData();  
   void setCustomerIdNum(int num);  
   void setLastName(char name[]);  
   void setFirstName(char name[]);  
   void setCreditLimit(double CL);  
};

//implementation section  
void Customer::displayCustomerData()  
{  
   cout<<"Customer ID Number "<<customerIdNum<<"'s first name is "<<firstName<<endl;  
   cout<<firstName<<"'s last name is "<<lastName<<endl;  
   cout<<"The credit limit for this customer is $"<<creditLimit<<endl;  
}  
void Customer::setCreditLimit(double CL)  
{  
   if( CL < 10000)  
     creditLimit = CL;  
   else  
     creditLimit = 9999;  
}  
void Customer::setLastName(char name[])  
{  
   strcpy(lastName, name);  
}  
void Customer::setFirstName(char name[])  
{  
   strcpy(firstName, name);  
}  
void Customer::setCustomerIdNum(int num)  
{  
   if ( customerIdNum < 10000)  
     customerIdNum = num;  
   else  
     customerIdNum = 9999;  
}  
void main()  
{  
  
  
  Customer aCustomer[5];  
   aCustomer[0].setCreditLimit(7000);  
   aCustomer.setLastName("Andrews");  
   aCustomer.setFirstName("Melissa");  
   aCustomer.setCustomerIdNum(754);  
   aCustomer.displayCustomerData();  
   getch();  
}

1. Unaitas Bank has decided to computerize its banking processes and one object needed is a BankAccount.
   * 1. Develop a struct so that it contains public data fields that hold the int account number and double account balance. ( 2 Marks)

Struct BankAccount{

Int account\_number;

Double account\_balance;

};

Write the main method to read in 10 bank account details from the keyboard and store in an array. Note that the accounts MUST be different ( 2 Marks)

#include<iostream>

#define MAX = 10

Struct BankAccount{

Int account\_number;

Double account\_balance;

}inputTimesArray[MAX];

int main ()

{

for(int n=0; n<=MAX; n++)

{

cout << “Enter Account Number”;  
 cin.getline(inputTimesArray[n].account\_number;);

cout << “Enter Account Balance”;  
 cin.getline(inputTimesArray[n].account\_balance;);

}

}

* + 1. How would your program in b above change if the number of bank accounts being read in are not known?

1. Convert the BankAccount data structure into a COMPLETE class with one constructor that creates an account with no minimum balance and another one that that initializes the minimum balance to Ksh. 5000, a destructor and three member functions, An enterAccountData()function that declares a local BankAccount object and prompts the user for values for each data field. Allow neither a negative balance nor a negative interest rate; continue to prompt the user until valid values are entered. The function returns a data-filled BankAccountobject to the calling function. A computeInterest() function that accepts a BankAccount parameter. Within the function, prompt the user for the number of years the account will earn interest. The function displays the ending balance of the account each year for the number of years entered based on the interest rate attached to the BankAccount. A main()function that declares a BankAccount object and continues to get BankAccount values from the user until the user enters a BankAccount with value 0 for the account number. For each BankAccount entered, display the BankAccount details and a list of future balances based on the term the user requests. stac
2. Clearly illustrate the distinction between the BankAccount structure and the BankAccount class above.
3. The only difference is if you don’t specify the visibility (public, private or protected) of the members, they will be public in the struct and private in the class. And the visibility by default goes just a little further than members: for inheritance if you don’t specify anything then the struct will inherit publicly from its base class:

1. Data items for a C++ program can be located in one of three areas: (1) stack, (2) heap or (3) global memory. For each of three, give short clear example C++ code that allocates a data item in that area. For each example, make clear what data item (variable etc) is allocated in the area.

## The Stack

What is the stack? It's a special region of your computer's memory that stores temporary variables created by each function (including the main() function). The stack is a "LIFO" (last in, first out) data structure, that is managed and optimized by the CPU quite closely. Every time a function declares a new variable, it is "pushed" onto the stack. Then every time a function exits, **all** of the variables pushed onto the stack by that function, are freed (that is to say, they are deleted). Once a stack variable is freed, that region of memory becomes available for other stack variables.

#include <stdio.h>

double multiplyByTwo (double input)

{

double twice = input \* 2.0;   
 return twice;   
 }   
  
int main (int argc, char \*argv[])   
{   
  
int age = 30;   
double salary = 12345.67;   
double myList[3] = {1.2, 2.3, 3.4};   
  
printf("double your salary is %.3f\n", multiplyByTwo(salary));   
  
return 0;   
}

double your salary is 24691.340

## The Heap

The heap is a region of your computer's memory that is not managed automatically for you, and is not as tightly managed by the CPU. It is a more free-floating region of memory (and is larger). To allocate memory on the heap, you must use malloc() or calloc(), which are built-in C functions. Once you have allocated memory on the heap, you are responsible for using free() to deallocate that memory once you don't need it any more. If you fail to do this, your program will have what is known as a **memory leak**. That is, memory on the heap will still be set aside (and won't be available to other processes). As we will see in the debugging section, there is a tool called valgrindthat can help you detect memory leaks.

Unlike the stack, the heap does not have size restrictions on variable size (apart from the obvious physical limitations of your computer). Heap memory is slightly slower to be read from and written to, because one has to use **pointers** to access memory on the heap. We will talk about pointers shortly.

#include <stdio.h>   
#include <stdlib.h>   
  
double \*multiplyByTwo (double \*input)

{   
  
 double \*twice = malloc(sizeof(double));   
 \*twice = \*input \* 2.0; return twice;   
 }   
 int main (int argc, char \*argv[])   
 {

int \*age = malloc(sizeof(int));   
\*age = 30;   
double \*salary = malloc(sizeof(double));   
\*salary = 12345.67;   
double \*myList = malloc(3 \* sizeof(double)); myList[0] = 1.2;   
  
myList[1] = 2.3; myList[2] = 3.4; double \*twiceSalary = multiplyByTwo(salary);   
  
printf("double your salary is %.3f\n", \*twiceSalary);   
 free(age);   
free(salary);   
free(myList);   
free(twiceSalary);   
return 0;   
}

Global memory is pre-allocated in a fixed memory block, or on the heap, depending on how it is allocated by your application:

byte x[10]; // pre-allocated by the compiler in some fixed memory block

byte \*y

main() {

y = malloc(10); // allocated on the heap

}