**FRE-GY 6883 Financial Computing**

**Midterm Exam Sample Questions**

Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_First Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_NYUID:\_\_\_ \_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_\_

**Q1: Change the implementation of the following Swap program from using Call by Pointer to Call by Reference**

|  |
| --- |
| SwapUtility.cpp |
| #include <iostream>  using namespace std;  class SwapUtility  {  private:  int\* const ptr1;  int\* const ptr2;  public:  SwapUtility() : ptr1(NULL), ptr2(NULL) {}  SwapUtility(int\* const ptr1\_, int\* const ptr2\_) : ptr1(ptr1\_), ptr2(ptr2\_) {}  **// Complete the Swap() function**  void Swap()  {  int tmp = \*ptr1;  \*ptr1 = \*ptr2;  \*ptr2 = tmp;  }  };  **// Complete the main function**  int main()  {  int a = 10, b = 20;  cout << "Before Swap()" << " a = " << a << " b = " << b << endl;  SwapUtility swap(&a, &b);  swap.Swap();  cout << "After Swap()" << " a = " << a << " b = " << b << endl;  return 0;  }  /\*  Before Swap() a = 10 b = 20  After Swap() a = 20 b = 10  \*/ |

**Q2. Specify the Output Results from the following program.**

|  |  |
| --- | --- |
| Program | Output Results |
| #include <iostream>  using namespace std;  class FRE6833  {  private:  int iID\_;  public:  FRE6833():iID\_(0)  { cout << "Default Constructor: iID = "  << iID\_ << endl;  }  FRE6833(int iID):iID\_(iID)  { cout << "Constructor with Paramter: iID = "  << iID\_ << endl;  }  FRE6833(const FRE6833 & course):iID\_(course.iID\_)  { cout << "Copy Constructor: iID = "  << iID\_ << endl; }  ~FRE6833()  { cout << "Destructor: iID = "  << iID\_ << endl;  }  int GetID() { return iID\_; }  };  FRE6833 Max(FRE6833 a, FRE6833 b)  {  if ( a.GetID() > b.GetID() )  return a;  else  return b;  }  void Q2()  { FRE6833 class1(1);  FRE6833 class2 = class1;  FRE6833 classX(5), classY(6);  FRE6833 classZ;  classZ = Max(classX, classY);  FRE6833 \*pClasses = new FRE6833[3];  delete [] pClasses;  }  int main(int argc, char \*arg[])  {  Q2();  return 0;  } | Constructor with Paramter: iID = 1  Copy Constructor: iID = 1  Constructor with Paramter: iID = 5  Constructor with Paramter: iID = 6  Default Constructor: iID = 0  Copy Constructor: iID = 6  Copy Constructor: iID = 5  Copy Constructor: iID = 6  Destructor: iID = 5  Destructor: iID = 6  Destructor: iID = 6  Default Constructor: iID = 0  Default Constructor: iID = 0  Default Constructor: iID = 0  Destructor: iID = 0  Destructor: iID = 0  Destructor: iID = 0  Destructor: iID = 6  Destructor: iID = 6  Destructor: iID = 5  Destructor: iID = 1  Destructor: iID = 1 |

**Q3. Complete the following function by displaying all the Price Points in the vector by using iterator.**

|  |
| --- |
| #include "Options05.h"  #include "BinModel02.h"  #include <iostream>  #include <cmath>  #include <vector>  using namespace std;  double EurOption::PriceByCRR(BinModel Model, double K)  {  double q=Model.RiskNeutProb();  vector<double> Price(N+1);  for (int i=0; i<=N; i++)  {  Price[i]=Payoff(Model.S(N,i),K);  }  for (int n=N-1; n>=0; n--)  {  for (int i=0; i<=n; i++)  {  Price[i]=(q\*Price[i+1]+(1-q)\*Price[i])  /(1+Model.GetR());  }  }  // Display all the Price Points in the vector by using iterator  vector<double>::iterator itr = Price.begin();  for ( ; itr != Price.end(); itr++ )  cout << \*itr << " ";  cout << endl;  return Price[0];  } |

**Q4.Change class and functions (including class member functions) to template class and function templates, so the class and functions could satisfy the main() function listed as the following.**

|  |  |
| --- | --- |
| BinModel02.h | Convert to template class and function templates |
| #include "BinModel02.h"  #include <iostream>  #include <cmath>  using namespace std;  class BinModel  { private:  double S0;  double U;  double D;  double R;  public:  double RiskNeutProb();  double S(int n, int i);  int GetInputData();  double GetR();  };  double BinModel::RiskNeutProb()  {  return (R-D)/(U-D);  }  double BinModel::S(int n, int i)  {  return S0\*pow(1+U,i)\*pow(1+D,n-i);  }  int BinModel::GetInputData()  {  cout << "Enter S0: "; cin >> S0;  cout << "Enter U: "; cin >> U;  cout << "Enter D: "; cin >> D;  cout << "Enter R: "; cin >> R;  cout << endl;  if (S0<=0.0 || U<=-1.0 ||  D<=-1.0 || U<=D || R<=-1.0)  { cout << "Illegal data ranges" << endl;  cout << "Terminating program" << endl;  return 1; }  if (R>=U || R<=D)  { cout << "Arbitrage exists" << endl;  cout << "Terminating program" << endl;  return 1; }  cout << "Input data checked" << endl;  cout << "There is no arbitrage" << endl << endl;  return 0;  }  double BinModel::GetR()  {  return R;  } | **template <typename T>**  class BinModel  { private:  **T S0;**  **T U;**  **T D;**  **T R;**  public:  **T** RiskNeutProb();  **T** S(int n, int i);  int GetInputData();  **T** GetR();  };  **template <typename T>**  **T BinModel<T>::RiskNeutProb()**  {  return (R-D)/(U-D);  }  **template <typename T>**  **T BinModel<T>::S(int n, int i)**  {  return S0\*pow(1+U,i)\*pow(1+D,n-i);  }  **template <typename T>**  **int BinModel<T>::GetInputData()**  {  //entering data  cout << "Enter S0: "; cin >> S0;  cout << "Enter U: "; cin >> U;  cout << "Enter D: "; cin >> D;  cout << "Enter R: "; cin >> R;  cout << endl;  //making sure that 0<S0, -1<D<U, -1<R  if (S0<=0.0 || U<=-1.0 || D<=-1.0 || U<=D  || R<=-1.0)  {  cout << "Illegal data ranges" << endl;  cout << "Terminating program" << endl;  return 1;  }  if (R>=U || R<=D)  {  cout << "Arbitrage exists" << endl;  cout << "Terminating program" << endl;  return 1;  }  cout << "Input data checked" << endl;  cout << "There is no arbitrage" << endl << endl;  return 0;  }  **template <typename T>**  **T BinModel<T>::GetR()**  {  return R;  } |
| Options04.h | Convert to function templates |
| include "BinModel02.h"  #include <iostream>  #include <cmath>  using namespace std;  int GetInputData(int& N, double& K)  { cout << "Enter steps to expiry N: "; cin >> N;  cout << "Enter strike price K: "; cin >> K;  cout << endl;  return 0;  }  double PriceByCRR(BinModel Model, int N, double K,  double (\*Payoff)(double z, double K))  { double q=Model.RiskNeutProb();  double Price[N+1];  for (int i=0; i<=N; i++)  { Price[i]=Payoff(Model.S(N,i),K); }  for (int n=N-1; n>=0; n--)  { for (int i=0; i<=n; i++)  {  Price[i]=(q\*Price[i+1]+(1-q)\*Price[i])  /(1+Model.GetR());  } }  return Price[0];  }  double CallPayoff(double z, double K)  { if (z>K) return z-K;  return 0.0;  }  double PutPayoff(double z, double K)  { if (z<K) return K-z;  return 0.0;  } | #include "BinModel02.h"  #include <vector>  #include <iostream>  #include <cmath>  using namespace std;  **template <typename T>**  **int GetInputData(int& N, T& K)**  {  cout << "Enter steps to expiry N: "; cin >> N;  cout << "Enter strike price K: "; cin >> K;  cout << endl;  return 0;  }  **template <typename T>**  **T PriceByCRR(BinModel<T> Model, int N, T K,**  **T (\*Payoff)(T z, T K))**  { **T q=Model.RiskNeutProb();**  vector<T> Price(N+1);  for (int i=0; i<=N; i++)  { Price[i]=Payoff(Model.S(N,i),K); }  for (int n=N-1; n>=0; n--)  { for (int i=0; i<=n; i++)  {  Price[i]=(q\*Price[i+1]+(1-q)\*Price[i])  /(1+Model.GetR());  } }  return Price[0];  }  **template <typename T>**  **T CallPayoff(T z, T K)**  { if (z>K) return z-K;  return 0.0; }  **template <typename T>**  **T PutPayoff(T z, T K)**  { if (z<K) return K-z;  return 0.0; } |
| Main using Template Class and Function Templates | |
| #include "BinModel02.h"  #include "Options04.h"  #include <iostream>  #include <cmath>  using namespace std;  int main()  {  BinModel<double> Model;  if (Model.GetInputData()==1) return 1;  double K; //strike price  int N; //steps to expiry  GetInputData<double>(N,K);  cout << PriceByCRR<double>(Model,N,K,CallPayoff) << endl;  GetInputData<double>(N,K);  cout << PriceByCRR<double>(Model,N,K,PutPayoff) << endl;  return 0;  } | |

Q5: Re-implement the class DefInt by using (1) Abstract Base Class, DefInt. (2) Create two derived classes, Trapezoid and Simposon. (3) The function double ByTrapezoid(int N) and double BySimpson(int N) will become private member functions, which are invoked in the function ByNumApprximation().

|  |  |  |
| --- | --- | --- |
| Original DefInt Class | Re-implement DefInt | |
| class DefInt  {  private:  double a,b;  double (\*f)(double x);  public:  DefInt(double a\_, double b\_, double (\*f\_)(double x))  {a=a\_; b=b\_; f=f\_;}  double ByTrapezoid(int N);  double BySimpson(int N);  };  double DefInt::ByTrapezoid(int N)  {  cout << "ByTrapezoid: ";  double h=(b-a)/N;  double Result=0.5\*f(a);  for (int n=1; n<N; n++) Result+=f(a+n\*h);  Result+=0.5\*f(b);  return Result\*h;  }  double DefInt::BySimpson(int N)  {  cout << "BySimpson: ";  double h=(b-a)/N;  double Result=f(a);  for (int n=1; n<N; n++) Result+=4\*f(a+n\*h  - 0.5\*h)+2\*f(a+n\*h);  Result+=4\*f(b-0.5\*h)+f(b);  return Result\*h/6;  }  double f(double x){return x\*x\*x-x\*x+1;} | class DefInt  {  protected:  int N;  double a,b;  double (\*f)(double x);  public:  DefInt(int N\_, double a\_, double b\_,  double (\*f\_)(double x)):N(N\_),a(a\_),b(b\_),f(f\_) {}  virtual ~DefInt() { }  virtual double ByNumApprximation() = 0;  };  **// Derived classes Trapezoid and Simpson:**  class Trapezoid: public DefInt  { private:  double ByTrapezoid(int N);  public:  Trapezoid(int N\_, double a\_, double b\_,  double (\*f\_)(double x)):DefInt(N\_, a\_, b\_, f\_) {}  ~Trapezoid() {}  virtual double ByNumApprximation() { return ByTrapezoid(N); }  };  class Simpson: public DefInt  { private:  double BySimpson(int N);  public:  Simpson(int N\_, double a\_, double b\_,  double (\*f\_)(double x)):DefInt(N\_, a\_, b\_, f\_) {}  ~Simpson() {}  virtual double ByNumApprximation() { return BySimpson(N); }  };  double Trapezoid::ByTrapezoid(int N)  { cout << "ByTrapezoid: ";  double h=(b-a)/N;  double Result=0.5\*f(a);  for (int n=1; n<N; n++) Result+=f(a+n\*h);  Result+=0.5\*f(b);  return Result\*h;  }  double Simpson::BySimpson(int N)  { cout << "BySimpson: ";  double h=(b-a)/N;  double Result=f(a);  for (int n=1; n<N; n++) Result+=4\*f(a+n\*h-0.5\*h)+2\*f(a+n\*h);  Result+=4\*f(b-0.5\*h)+f(b);  return Result\*h/6;  } | |
| **Q6-Q8 are based on the following classes and independent functions:** | | |
| Solver03.h | | |
| #pragma once  template<typename Function>  double SolveByBisect(Function\* Fct, double Tgt, double LEnd, double REnd, double Acc)  { double left=LEnd, right=REnd, mid=(left+right)/2;  double y\_left=Fct->Value(left)-Tgt, y\_mid=Fct->Value(mid)-Tgt;  while (mid-left>Acc)  {  if ((y\_left>0 && y\_mid>0)||(y\_left<0 && y\_mid<0))  {left=mid; y\_left=y\_mid;}  else right=mid;  mid=(left+right)/2;  y\_mid=Fct->Value(mid)-Tgt;  }  return mid;  }  template<typename Function>  double SolveByNR(Function\* Fct, double Tgt, double Guess, double Acc)  { double x\_prev=Guess;  double x\_next=x\_prev -(Fct->Value(x\_prev)-Tgt)/Fct->Deriv(x\_prev);  while (x\_next-x\_prev>Acc || x\_prev-x\_next>Acc)  {  x\_prev=x\_next;  x\_next=x\_prev-(Fct->Value(x\_prev)-Tgt)/Fct->Deriv(x\_prev);  }  return x\_next;  } | | |
| Bond.cpp | | |
| #include "Solver03.h"  #include <map>  #include <vector>  #include <cmath>  #include <iostream>  #include <iomanip>  using namespace std;  class Bond  {  private:  double F; //face value  int T; //maturity time  double yield;  map<int, double> couponMap;  public:  Bond(double F\_, int T\_, const vector<double>& C\_, const vector<int>& t\_):F(F\_),T(T\_),yield(0.0)  {  // Asssume vector C\_ and t\_ have same size  for (unsigned int i = 0; i < C\_.size(); i++)  couponMap[t\_[i]] = C\_[i];  }  double BondValue();  double BondDeriv();  double GetF() const { return F; }  int GetT() const { return T; }  double GetYield() const { return yield; }  void SetYield(double y) { yield = y; }  map<int, double>& GetCouponMap() { return couponMap; }  }; | | |

**Q6. Complete the implementation of BondValue() and BondDeriv() for the Bond class with the same formula we used in 2nd homework assignment for Nonlinear Solvers.**

|  |
| --- |
| double Bond::BondValue()  {  double P = 0;  for (map<int, double>::iterator itr = couponMap.begin(); itr != couponMap.end(); itr++)  P += itr->second \* exp(-yield \* itr->first);  P += F \* exp(-yield \* T);  return P;  }  double Bond::BondDeriv()  {  double D = 0;  for (map<int, double>::iterator itr = couponMap.begin(); itr != couponMap.end(); itr++)  D += -itr->second \* itr->first \* exp(-yield \* itr->first);  D += -F \* T \* exp(-yield \* T);  return D;  } |

**Q7. Complete the implementation of the following class Intermediary, as well as the corresponding codes in main() to use it.**

|  |
| --- |
| class Intermediary  {  private:  Bond \* ptr;  public:  Intermediary(Bond\* ptr\_) : ptr(ptr\_) {}  **double Value(double y)**  **{**  ptr->SetYield(y);  return ptr->BondValue();  **}**  **double Deriv(double y)**  **{**  ptr->SetYield(y);  return ptr->BondDeriv();  **}**  }; |
| int main()  {  double F=100.0; //face value  int T=3; //maturity time  vector<double> C; //coupons  C.push\_back(1.2); C.push\_back(1.2); C.push\_back(1.2);  vector<int> t; //coupon times  t.push\_back(1); t.push\_back(2); t.push\_back(3);  Bond MyBond(F,T,C,t);  double P=98.56;  double Acc=0.0001;  double y;  double LEnd = 0.0;  double REnd = 1.0;    **// Calculate the bond yield by bisection method**  Intermediary intermediary(&MyBond);  y=SolveByBisect<Intermediary>(&intermediary, P, LEnd, REnd, Acc);  MyBond.SetYield(y);  cout << setiosflags(ios::fixed) << setprecision(4);  cout << "P = " << P << endl;  cout << MyBond << endl;  return 0;  }  /\*  P = 98.5600  F = 100.0000  T = 3  Year 1 Coupon: 1.2000  Year 2 Coupon: 1.2000  Year 3 Coupon: 1.2000  yield = 0.0168  \*/ |

**Q8. Complete the following operator overloading to print MyBond object in main() function shown above.**

|  |
| --- |
| ostream& operator << (ostream& out, Bond& bond)  {  out << "F = " << bond.GetF() << endl;  out << "T = " << bond.GetT() << endl;    for (map<int, double>::iterator itr = bond.GetCouponMap().begin(); itr != bond.GetCouponMap().end(); itr++)  {  out << "Year " << itr->first << " Coupon: " << itr->second << endl;  }  out << endl;  out << "yield = " << bond.GetYield() << endl;  return out;  } |