Q-1 10+10+10+10

1. Consider a BankAccount class has minbal (Minimum Balance) and curbal (Current Balance) as float data members. A member function void BankAccount::withdraw(float amount) has to reduce the curbal by amount but maintain it above the minbal. In exceptional case, withdraw throws ‘S’ as an exception of the char type. Write code for the withdraw member function and sample instructions to demonstrate the handling of such exception.

void BankAccount::withdraw(float amount)

{

float rembal = curbal - amount;

if(rembal > minbal)

{

curbal = rembal;

}

else

{

throw 'S';

}

}

main ...

BankAccount acc(...);

try

{

acc.withdraw(24000f);

}

catch(char ex)

{

if (ex == 'S')

{

cout << "low balance" << endl ;

}

}

1. For hierarchy of shapes, assume **rectangle** class has a method draw given (next to question statement). Define (write code) the similar methods for **square** and **triangle** classes. *[Note: data members of rectangle class are point2d topleft, int length, int width and that of triangle are v1, v2 and v3 of type point2d]*

void rectangle::draw()

{

point2d p1(this->topleft);

point2d p2(this->topleft.getx(), this->topleft.gety()+this->length);

point2d p3(this->topleft.getx()+this->width, this->topleft.getyy());

point2d p4(this->topleft.getx()+this->width, this->topleft.gety()+length);

drawline(p1, p2);

drawline(p1, p3);

drawline(p2, p4);

drawline(p3, p4);

}

For square, there are three easy possibilities

1. We don’t need to do anything as square is inheriting rectangle in such a way that draw square properly.
2. Following simple solution

void square::draw()

{

rectangle::draw();

}

1. Following same solution

void square::draw()

{

point2d p1(this->topleft);

point2d p2(this->topleft.getx(), this->topleft.gety()+this->length);

point2d p3(this->topleft.getx()+this->width, this->topleft.getyy());

point2d p4(this->topleft.getx()+this->width, this->topleft.gety()+length);

drawline(p1, p2);

drawline(p1, p3);

drawline(p2, p4);

drawline(p3, p4);

}

For triangle, following are the two best ways

1. Following simple solution

void triangle::draw()

{

drawline(v1, v2);

drawline(v2, v3);

drawline(v3, v1);

}

1. Following same solution

void triangle::draw()

{

point2d p1(this->v1);

point2d p1(this->v2);

point2d p1(this->v3);

drawline(p1, p2);

drawline(p2, p3);

drawline(p3, p1);

}

1. Given the following definition of UF function, write recursive function to compute and return UF of two integers passed as its arguments.

long uf(long a, long b)

{

if(b==0)

{

return a;

}

else

{

long t = uf(b, a%b);

return t;

}

}

1. Write a recursive function void printOctal(int) to print the octal number equivalent to its integer argument. Conversion of a number from decimal system to octal is: successive divisions by eight ( 8 ) until the quotient is smaller than 8 and then joining it with the remainders in reverse order of successive divisions already done.

void printOctal(int n)

{

int q = n/8;

int r = n%8;

if(q < 8)

{

cout << q;

}

else

{

printOctal(q);

}

cout << r;

}

Q-2 10+10+10+10

Using polymorphism, you are required to implement the inheritance hierarchy shown in figure to compute percent marks of various types of **Student**s.

There is one base class (Student) to store common data, and three derived classes that divide the set of students into three categories: English students (EngStudent), Computer Science students (CompStudent), and Math students (MathStudent). All the data members of the Student class should be declared as *protected*. Note that the member variables of each of the three derived classes are not shown in the above class diagram. Each derived class will have appropriate number of *private* data members to store the marks of different assessments applicable for a particular student. Assessments and their weightages for the three categories of students are as follows:

* **English** – Attendance = 10%, Report = 30%, Midterm = 30%, Final Exam = 30%
* **Computer Science** – Project = 25%, Midterm = 35%, Final Exam = 40%
* **Math** – Quiz Average\* = 25%, Homeworks = 25%, Final Exam = 50%

Note the following important points:

1. To save time, you should not write the details of getters/setters, constructors and other typical implementation of member functions
2. The marks of all assessments are integers given out of 100
3. Student class must define float percentMarks() as pure virtual
4. You may have to create a class ExamSys to aggregate Students
5. You MUST have to write driver code (or main function) for sufficient hard coded student’s data

class Student

{

protected:

int rollno;

string name;

double totalMarks;

char grade;

public:

// typical gettes and setters

// default contructor

// first two data member constructor

string toString()

{

//typical implementation

}

// other behaviour

virtual float percentMarks() =0;

};

class EngStudent : public Student

{

private:

int att;

int rep;

int mid;

int fin;

public:

// typical gettes and setters

// default contructor

EngStudent(int ro, string nm, int at, int rp, int md, int fn):Student(ro, nm)

{

//typical implementation

}

string toString()

{

//typical implementation

}

float percentMarks()

{

return att\*0.1f + rep\*0.3f + mid\*0.3f + fin\*0.3f ;

}

};

class MathStudent : public Student

{

private:

int qiz;

int hmw;

int fin;

public:

// typical gettes and setters

// default contructor

MathStudent(int ro, string nm, int qz, int hw, int fn):Student(ro, nm)

{

//typical implementation

}

string toString()

{

//typical implementation

}

float percentMarks()

{

return qiz\*0.25f + hmw\*0.25f + fin\*0.5f ;

}

};

class ExamSys

{

private:

Student \*ar

int capacity;

int count;

public:

ExamSys()

{

// typical dynamic list implementation

}

~ExamSys()

{

// typical dynamic list implementation

}

void addStudent(Student \*st)

{

// typical dynamic list implementation

}

void display()

{

int j = 0;

while(j < count)

{

cout << ar[j]->toString() << " " << ar[j]->percentMarks();

}

}

};

main ...

ExamSys es(50);

MathStudent m1(123, "jalil khan", 76, 59, 86);

MathStudent m2(...);

MathStudent m3(...;

EngStudent e2(836, "rifat khan", 65, 76, 78, 80);

EngStudent e3(...);

es.addStudent(m1);

es.addStudent(m2);

es.addStudent(m3);

es.addStudent(e3);

es.addStudent(e2);

es.display();

Q-3 10+10+10+10

1. Write output of the following code segments.

|  |  |
| --- | --- |
| namespace ns1  {  int ns1i1;  namespace ns2  {  int ns1i1;  int ns2i1;  namespace ns3  {  int ns1i1;  int ns3i1;  }  }  } | int main(void)  {  ns1::ns2::ns3::ns1i1 = 10;  ns1::ns2::ns1i1 = 20;  ns1::ns1i1 = 30;  ns1::ns2::ns3::ns3i1 = 40;  cout << ns1::ns2::ns3::ns3i1 << endl;  cout << ns1::ns1i1 << endl;  cout << ns1::ns2::ns1i1 << endl;  cout << ns1::ns2::ns3::ns1i1 << endl;  return -1;  } |
| class A {  public:  int a;  A(int x):a(x){}  };  class B : public A {  public:  int a;  B(int x):a(x), A(3){}  int bf(){return A::a;}  };  class C : public B {  public:  int a, b;  C(int x, int y, int z):B(y)  {b=x+y; a=z; A::a=y;}  int af(){return A::a;}  int bf(){return B::a;} | int cf(){return a;}  int df(){return a;}  };  int main()  {  A u(7);  B t(4);  C s(3, 8, 1);  B \*p = &s;  cout << u.a << endl;  cout << t.a << endl;  cout << t.bf() << endl;  cout << s.b << endl;  cout << s.bf() << endl;  cout << s.df() << endl;  cout << p->a << endl;  cout << p->bf() << endl;  } |

40

30

20

10

7

4

3

11

8

1

8

8

1. Explain diamond problem and its resolution.

Two copies of the grandparent will be included in grandchild due to a diamond like hierarchy and it is resolved using virtual inheritance.

1. Describe the use of abstract classes?

These are used to implement STANDARDS.

1. How late binding is achieved in CPP?

It is achieved through function pointers.

Q-4 10+10+10+10

In mathematics, a polynomial is an expression of finite length constructed from variables and constants, e.g. - 2x3 - 5x2 + 1 is a polynomial of of order ***three*** of variable ***x*** having ***four*** (one is not mentioned explicitly) terms. Each term in polynomial contains a coefficient and an exponent. In term -5x2 has coefficient the -5 and exponent the 2 and coefficient of x1 is *zero*. Consider you are provided the class Term as described below.

class Term

{

private:

float coef;

int expo;

public:

float getCoefficinet() const;

Don’t write code for this class, use this description to define Polynomial class

int getExponent() const;

void setCoefficinet(float);

void setExponent(int);

void setTerm(float, int);

Term();

Term(float, int);

string toString() const;

Term parseTerm(string);

};

1. You have to describe class Polynomial comprehensively with data members as 1) int order 2) char variable, and 3) Term \* polyterms

Hint: order and polyterms may be used as basic necessity of a dynamic list

class Polynomial

{

protected:

int order;

char variable;

Term \* polyterms

public:

int getOrder() const;

void setOrder(int);

char getVariable() const;

void setPolynomial(int, char);

void setVariable(char);

float getTerm(int) const; // param is expo, return is coef

void setTerm(float, int); // params are coef and expo

Polynomial(int);

Polynomial(int, char);

Polynomial(Polynomial &);

~Polynomial();

string toString() const;

Polynomial parsePolynomial(string);

float evaluate(double);

void sort();

static Polynomial getSort(const Polynomial &);

// more function for behaviour of Polynomial type

Polynomial & operator=(const Polynomial &);

Polynomial operator+(const Polynomial &);

Polynomial operator-(const Polynomial &);

Polynomial operator\*(const Polynomial &);

Polynomial operator/(const Polynomial &);

// operations with number on lhs & rhs

Polynomial operator==(const Polynomial &);

Polynomial operator!=(const Polynomial &);

Polynomial operator<(const Polynomial &);

Polynomial operator<=(const Polynomial &);

Polynomial operator>(const Polynomial &);

Polynomial operator>=(const Polynomial &);

// possibly other streaming, unary and etc opertions

// subscript [], parenthesis () and related operators

// etc

};

1. Define parameterized constructor, destructor and toString() member function.

Polynomial::Polynomial(int od, char vr)

{

int order = od;

char variable = vr;

polyterms = new Term[od+1]; // with exception

for(int j=0; j<=order; j++)

{

polyterms[j].setTerm(0, j);

}

}

Polynomial::~Polynomial()

{

delete []polyterms;

}

string Polynomial::toString()

{

ostringstream oss;

for(int j=order; j>=0; j--)

{

if(this->polyterms[j].coef != 0)

{

if(this->polyterms[j].coef > 0)

{

oss << "+" ;

}

oss << this->polyterms[j].coef;

oss << this->variable;

oss << "^";

oss << this->polyterms[j].expo;

}

}

if(oss[0] == '+')

{

oss.remove(0);

}

return oss.str();

}

1. Define overloaded \* operation to multiply to polynomials

Polynomial Polynomial::operator\*(const Polynomial &rhs)

Polynomial Polynomial::operator\*(const Polynomial &rhs)

{

if(this->variabe != rhs.variable)

{

throw string("variable mismatch");

}

Polynomial p1(\*this);

Polynomial p2(rhs);

p1.sort();

p2.sort();

Polynomial rv(this->order+rhs.order+1, this->variable);

for(int j=p1.order; j>=0; j--)

{

for(int k=p2.order; k>=0; k--)

{

rv.setTerm(p1.getTerm(j) \* p2.getTerm(k), j+k);

}

}

return rv;

}

1. Define float Polynomial::evaluate(double) members function. Hint: argument of function is the value of polynomial variable for evaluation.

float Polynomial::evaluate(double v)

{

float rv = 0.0f;

for(int j=p1.order; j>=0; j--)

{

rv += this->p1.getTerm(j) \* pow(v, j);

}

return rv;

}

--- Best of Luck ---