Maciej Sobieraj

Lecture 14

Outline

1. Examples
    1. Inheritance
        1. **Example 1**
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 1

- Write four classes to implement the printing of different kinds of trees (they should consist of very simple ASCII characters, i.e. '*', '/, and '\').
  - One of the classes is a base for the other three. In the base class, you should create a draw method – it's the only method of this class and it's a virtual method. Next, in the derived classes, implement this method with three different versions of drawing.
  - In the main function, create a table of three pointers to objects of the base class. Then create three objects of different kinds of trees and assign pointers to them to cells of the previously created table. After this, in the for loop, iterate over the table and call the draw method from every element of the table.

Example 1

```
#include <iostream>

using namespace std;


class BaseTree
{
public:
 virtual void draw() {};
};
```

Example 1

```
class SimpleTree :public BaseTree
{

public:

 void draw()
 {
  cout << " /\\\n";
  cout << "//\\\\\\n";
 }

};
```

```
class StarTree :public BaseTree
{

public:

 void draw()
 {
  cout << " /\\\n";
  cout << "/**\\\n";
 }

};
```

```
class PlusTree :public BaseTree
{

public:

 void draw()
 {
  cout << " /\\\n";
  cout << "/++\\\n";
 }

};
```

Example 1

```
int main()
{
 BaseTree *trees[3];

 SimpleTree simpleTree;
 StarTree starTree;
 PlusTree plusTree;

 trees[0] = &simpleTree;
 trees[1] = &starTree;
 trees[2] = &plusTree;

 for (int i = 0; i < 3; i++)
 {
  cout << "Drawing " << i + 1 << ": \n";
  trees[i]->draw();
 }
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. **Example 2**
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 2

- Let's assume you write a code for the game of checkers (aka draughts).
  - There are two types of pieces in this game: men and kings.
  - You need to implement classes (one abstract for the pieces and two separate classes for the men and the kings) to check if the moves are correct.
    - Men can only move one field forward along diagonals.
    - In international checkers, kings can move any distance forward and backward along unblocked diagonals. To simplify your program, assume that the board is empty and you only have to check one piece at a time.
  - The starting position for one of the pieces is the b1 field. Call a check method from the pointer to the base (Piece) class.

Example 2

```
#include <iostream>
#include <string>

using namespace std;

class Piece
{
public:
 virtual bool isMovePossible(string from, string to) { return false; };
protected:
 static bool isnumberInRange(int number)
 {
  return number <= '8' && number >= '1';
 }

 static bool isLetterInRange(char letter)
 {
  return letter <= 'h' && letter >= 'a';
 }

};
```

Example 2

```
class King :public Piece
{
public:
 bool isMovePossible(string from, string to);
};

bool King::isMovePossible(string from, string to)
{
 bool isPossible = false;
 if (from.length()==2 && to.length()==2)
 {
  char letterFrom = from[0];
  char numberFrom = from[1];
  char letterTo = to[0];
  char numberTo =to[1];
  if (abs(letterFrom - letterTo) == abs(numberFrom- numberTo) )
  {
   if (isnumberInRange(numberFrom) && isLetterInRange(letterFrom))
   {
    if (isnumberInRange(numberTo) && isLetterInRange(letterTo))
    {
     isPossible = true;
    }
   }
  }
 }
 return isPossible;
}
```

Example 2

```
class Man:public Piece
{
public:
 bool isMovePossible(string from, string to)
 {
  bool isPossible = false;
  if (from.length() == 2 && to.length() == 2)
  {
   char letterFrom = from[0];
   char numberFrom = from[1];
   char letterTo = to[0];
   char numberTo = to[1];
   if (abs(letterFrom - letterTo) == abs(numberFrom - numberTo) && abs(numberFrom - numberTo) ==
   {
    if (isnumberInRange(numberFrom) && isLetterInRange(letterFrom))
    {
     if (isnumberInRange(numberTo) && isLetterInRange(letterTo))
     {
      isPossible = true;
     }
    }
   }
  }
  return isPossible;
 }
};
```

Example 2

```
int main()
{
 Piece *pieces[3];
 pieces[0] = new Man();
 pieces[1] = new King();
 pieces[2] = new Man();
 cout.setf(ios::boolalpha);
 cout << pieces[0]->isMovePossible("b1", "c2") << endl;
 cout << pieces[1]->isMovePossible("b1", "d3") << endl;
 cout << pieces[2]->isMovePossible("b1", "d3") << endl;
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. **Example 3**
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 3

- Write a hierarchy of two classes: base class IPAddress and derived class IPAddressChecked.
  - The first class holds the value of an IP address and the derived class has one additional Boolean value (set to true if the IP is correct, otherwise set to false).
    - Both classes have a constructor, a copy constructor and the print method.
  - The print method in the derived class should also print the value of its Boolean (Correct/Not Correct).
  - Use the methods of the base class in the derived one.
  - Write some test code to get input from the user.
  - Create one IPAddress object and two IPAddressChecked objects. Print the values for all three objects.

Example 3

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

class IPAddress
{
public:
 IPAddress(const string address) : address(address) {}
 IPAddress(const IPAddress& other) : address(other.address) {}
 void print ()
 {
  cout << endl << address;
 }
protected:
 string address;
};
```

Example 3

```
class IPAddressChecked :public IPAddress
{
public:
 IPAddressChecked(const string& address) : IPAddress(address), isCorrect(check()) {}
 IPAddressChecked(const IPAddress& other) : IPAddress(other), isCorrect(check()) {}
 bool check()
 {
  stringstream splited(address);
  string s;
  int partsCount = 0;
  while (std::getline(splited, s, '.'))
  {
   if (3 < s.length() || s.length() < 1)
    return false;
   if (partsCount > 4)
    return false;
   for (int i = 0; i < s.length(); i++)
   {
    if (!isdigit(s[i]))
     return false;
   }
   int partValue = atoi(s.c_str());
   if (partValue > 255)
    return false;
   partsCount++;
  }
  if (partsCount != 4)
   return false;
  return true;
 }
```

Example 3

```
 void print()
 {
  IPAddress::print();
  cout << (isCorrect ? " - Correct" : " - Not Correct");
 }
protected:
 bool isCorrect;
};
```

Example 3

```
int main()
{
 string inputData;

 cout << "Input first IP address" << endl;
 cin >> inputData;
 //inputData = "1.2.3.4";//LEFT intentionally - for clarification
 IPAddress firstIP(inputData);

 cout << "Input second IP address" << endl;
 cin >> inputData;
 //inputData = "999.29.29.29";
 IPAddressChecked secondIP(inputData);

 cout << "Input third IP address" << endl;
 cin >> inputData;
 //inputData = "199.29.29.29";
 IPAddressChecked thirdIP(inputData);


 firstIP.print();
 secondIP.print();
 thirdIP.print();
 cout << endl;
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. **Example 4**
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 4

- In one of the network programs you need to store the IP addresses of a computer and other devices.
  - Create a class describing Network with an array of IPAddress. Implement methods to:
    - add one IPAddress to Network;
    - print all addresses in Network.
  - Create two Network objects and five IPAddress objects. One IPAddress should be placed in both networks.
  - Get five addresses from the user.
  - Print both networks.

Example 4

```
#include <iostream>
#include <string>

using namespace std;

class IPAddress
{
public:
 IPAddress() : address("") {}
 IPAddress(const string address) : address(address) {}
 IPAddress(const IPAddress& other) : address(other.address) {}
 IPAddress& operator=(const IPAddress& source)
 {
  if (this == &source)
   return *this;
  address = source.address;
  return *this;
 }

 string getAddress()
 {
  return address;
 }
 void setAddress(string address)
 {
  this->address=address;
 }
 void print()
 {
  cout << address << endl;
 }
private:
 string address;
};
```

Example 4

```
class IPNetwork
{
public:
 IPNetwork(const string& name) : name(name), addressCount(0)
 {}
 void addAddress(IPAddress address)
 {
  addresses[addressCount] = address;
  addressCount++;
 }

 void print()
 {
  cout << name << ":" << endl;
   addresses[i].print();
 }
private:
 string name;
 int addressCount;
 IPAddress addresses[255];
 //student can do it with dynamic memory allocation ie. IPAddress *addresses;
 //or with vector (not mentioned in this course)
};
```

Example 4

```
int main()
{
 string inputData;
 IPNetwork network1("Network 1");
 IPNetwork network2("Network 2");
 cout << "Input 1st IP address" << endl;
 cin >> inputData;
 //inputData = "1.1.1.1";//LEFT intentionally - for clarification
 network1.addAddress(IPAddress(inputData));

 cout << "Input 2nd IP address" << endl;
 cin >> inputData;
 //inputData = "2.2.2.2";
 network1.addAddress(IPAddress(inputData));

 cout << "Input 3rd IP address" << endl;
 cin >> inputData;
 //inputData = "3.3.3.3";
 IPAddress address(inputData);
 network1.addAddress(address);
 network2.addAddress(address);

 cout << "Input 4th IP address" << endl;
 cin >> inputData;
 //inputData = "4.4.4.4";
 network2.addAddress(IPAddress(inputData));

 cout << "Input 5th IP address" << endl;
 cin >> inputData;
 //inputData = "5.5.5.5";
 network2.addAddress(IPAddress(inputData));

 network1.print();
 network2.print();

 cout << endl;
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. **Example 5**
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 5

- Below is a recipe for constructing a primitive car. All class names start with a capital letter.
  - Compose a Car from some objects, you should use: one Engine, four Wheels, one Chassis, ten Lights, and one Body. Feel free to define the attributes of these classes.
  - Every class (including Car) must have a print method. In the print method of Car, you just need to call the print method from all the objects.
  - Create one car and call a print method.

Example 5

```
#include <iostream>
#include <string>

using namespace std;
#define className(x) #x
#define NUMBER_OF_WHEELS 4
#define NUMBER_OF_LIGHTS 10

class Engine
{
public:
 Engine() {}
 Engine(const string& type) : type(type) {}
 Engine(const Engine& source) : type(source.type) {}
 void print() const
 {
  cout << className(Engine) << ": " << type << endl;
 }
private:
 string type;
};
```

Example 5

```
class Wheel
{
public:
 Wheel(){}
 Wheel(const string& type) : type(type) {}
 Wheel(const Wheel& source) : type(source.type) {}
 void print() const
 {
  cout << className(Wheel) << ": " << type << endl;
 }
private:
 string type;
};

class Chassis
{
public:
 Chassis() {}
 Chassis(const string& type) : type(type) {}
 Chassis(const Chassis& source) : type(source.type) {}
 void print() const
 {
  cout << className(Chassis) << ": " << type << endl;
 }
private:
 string type;
};
```

Example 5

```
class Light
{
public:
 Light(){}
 Light(const string& type) : type(type) {}
 Light(const Light& source): type(source.type) {}
 void print() const
 {
  cout << className(Light) << ": " << type << endl;
 }
private:
 string type;
};
class Body
{
public:
 Body() {}
 Body(const string& type) : type(type) {}
 Body(const Body& source) : type(source.type) {}
 void print() const
 {
  cout << className(Body) << ": " << type << endl;
 }
private:
 string type;
};
```

Example 5

```
class Car
{
public:
 Car(const Engine& engine, const Chassis& chassis, const Body& body,
  Wheel wheels[NUMBER_OF_WHEELS], Light lights[NUMBER_OF_LIGHTS])
  : engine(engine), chassis(chassis), body(body)
 {
  for (int i = 0; i < NUMBER_OF_WHEELS; i++)
   this->wheels[i] = wheels[i];
  for (int i = 0; i < NUMBER_OF_LIGHTS; i++)
   this->lights[i] = lights[i];
 }

 void print() const
 {
  engine.print();
  for (int i = 0; i < NUMBER_OF_WHEELS; i++)
   wheels[i].print();
  chassis.print();
  for (int i = 0; i < NUMBER_OF_LIGHTS; i++)
   lights[i].print();
  body.print();
 }

private:
 Engine engine;
 Wheel wheels[NUMBER_OF_WHEELS];
 Chassis chassis;
 Light lights[NUMBER_OF_LIGHTS];
 Body body;
};
```

Example 5

```
int main()
{
 Engine engine("1.0");;
 Chassis chassis("Normal");
 Body body("Black");
 Light lights[]={ Light("Type 1"), Light("Type 1"),
  Light("Type 2"), Light("Type 2"),
  Light("Type 3"), Light("Type 3"),
  Light("Type 4"), Light("Type 4"),
  Light("Type 5"), Light("Type 5")
 };
 Wheel wheels[] = { Wheel("16inches"), Wheel("16inches"),
  Wheel("16inches"), Wheel("16inches") };

 Car car(engine, chassis, body, wheels, lights);
 car.print();

 cout << endl;
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. **Example 1**
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 1

- Improve this simple program so that it prints a message when the user inputs invalid data. Add code to throw the exception manually using the throw keyword when a user inputs 0.

```
#include <iostream>
using namespace std;
int main(void) {
 int a = 8, b = 0, c = 0;
 cin >> b;
 try {
  //Your code here
  c = a / b;
 }
 // Your code here
 cout << c << endl;
 return 0;
}
```

Example 1

```
#include <iostream>
using namespace std;
const int DivideByZero = 111;
int main(void) {
 int a = 8, b = 0, c = 0;
 cin >> b;
 try {
  if (b == 0)
  {
   throw DivideByZero;
  }
  c = a / b;
 }
 catch(int ex)
 {
  if (ex==DivideByZero)
   cout << "Your input is not valid, you can't divide by zero." << endl;
 }

 cout << c << endl;
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. **Example 2**
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 2

- Insert a try-catch block into this program to solve potential problems with arithmetic operations. Remember to inform the user of any exceptions in simple words.

```
#include <iostream>
using namespace std;
int main(void) {
 int a = 0, b = 0, c = 0;
 cin >> b;
 cin >> a;
 c = a / b;
 cout << c << endl;
 return 0;
}
```

Example 2

```
#include <iostream>
using namespace std;
const int DivideByZero = 111;
int main(void) {
 int a = 0, b = 0, c = 0;
 cin >> b;
 cin >> a;
 try
 {
  if (b == 0)
  {
   throw DivideByZero;
  }
  c = a / b;
  cout << c << endl;
 }
 catch(int ex)
 {
  if (ex==DivideByZero)
   cout << "Your input is not valid, you can't divide by zero." << endl;
 }
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. **Example 3**
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 3

- Try to change the program below (both the div and main functions).
  - Insert a throw and a try-catch block instead of a Boolean value and res parameter (change it to a function return value). Remember to inform the user of any exceptions in simple words.
  - Try to make two versions of your code, one that catches the division by zero exception, and another that checks arguments and then throws an exception (when the argument isn't proper).

Example 3

```
#include <iostream>

using namespace std;
const int DivideByZero = 111;
using namespace std;
float internaldiv(float arg1, float arg2)
{
 if (0==arg2)
  throw DivideByZero;
 return arg1 / arg2;
}

bool div(float &res, float arg1, float arg2) {
 if(arg2 == 0.0)
  return false;
 internaldiv(arg1, arg2);
 return true;
}

int main(void) {
 float r, a, b;
 while(cin >> a) {
  cin >> b;
  if(div(r,a,b))
   cout << r << endl;
  else
   cout << "Are you kidding me?" << endl;

 }
 return 0;
}
```

Example 3

```
#include <iostream>
using namespace std;
const int DivideByZero = 111;
using namespace std;
float internaldiv(float arg1, float arg2)
{
 if (0==arg2)
  throw DivideByZero;
 return arg1 / arg2;
}

float divv1(float arg1, float arg2) {
 try
 {
  return internaldiv(arg1, arg2);
 }
 catch(int ex)
 {
  throw ex;
 }

}
```

Example 3

```
int main2(void) {
 float r, a, b;
 while (cin >> a) {
  cin >> b;
  try {
   r = divv1(a, b);
   cout << r << endl;
  }
  catch (int ex)
  {
    cout << "Are you kidding me?" << endl;
    if (ex == DivideByZero)
     cout << "Your input is not valid, you can't divide by zero." << endl;
  }
 }
 return 0;
}
```

Example 3

```
float divv2(float &res, float arg1, float arg2) {
 if (arg2 == 0.0)
  throw DivideByZero;
 return internaldiv(arg1, arg2);
}

int main(void) {
 float r, a, b;
 while (cin >> a) {
  cin >> b;
  try {
   r = divv1(a, b);
   cout << r << endl;
  }
  catch (int ex)
  {
   cout << "Are you kidding me?" << endl;
   if (ex==DivideByZero)
    cout << "Your input is not valid, you can't divide by zero." << endl;
  }
 }
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. **Example 4**
        5. Example 5
    3. Operators
        1. Example 1
        2. Example 2

Example 4

- Write a program that calculates the area of some simple figures (i.e. a square and a rectangle). One function per figure.
  - Check if the function arguments are greater than 0 – if they aren't, throw an exception.
  - This program should use your own exceptions to communicate with a higher-level code.
  - Add some attribute in your exception to pass a message to the user.

Example 4

```
#include <iostream>

using namespace std;
//add your own exception class here
//add functions code here
int main(void) {
 float a, b, r;
 cin >> a;
 cin >> b;
 try
 {
  float rsquare = square_area(a);
  float rrectangle = rectangle_area(a,b);
  cout << rsquare << endl << rrectangle << endl;
 }
 //add a suitable catch block here
 return 0;
}
```

Example 4

```
#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;
class AreaException:public runtime_error
{
public:
 AreaException(const char * message): runtime_error(message){}
};

float square_area(float x)
{
 if (!(x > 0))
  throw AreaException("Your input is not valid, area can't be negative.");
 float result = x*x;
 return result;
}

float rectangle_area(float a, float b)
{
 if (!(a > 0))
  throw AreaException("Your input is not valid, area can't be negative.");
 float result = a*b;
 return result;
}
```

Example 4

```
int main(void) {
 float a, b, r;
 cin >> a;
 cin >> b;
 try
 {
  float rsquare = square_area(a);
  float rrectangle = rectangle_area(a, b);
  cout << rsquare << endl << rrectangle << endl;
 }
 catch(AreaException ex)
 {
  cout << ex.what() << endl;
 }
 //add a suitable catch block here
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. **Example 5**
    3. Operators
        1. Example 1
        2. Example 2

Example 5

- Write a class that holds a (2×2) matrix, and add two methods to work with files:
  - one method to load the matrix from a file (in any format)
  - and one method to save the matrix to a file (in the same format).
- Add code to handle exceptional situations (file not found and no rights to file), print a message and re-throw an exception.
- Add a try-catch block in the proper places. Simulate both situations handled (try to load a nonexistent file, and try to save a file in a path where you have no proper righ

Example 5

```
#include <iostream>
#include <fstream>
#include <stdexcept>
#include <string>

using namespace std;

class Matrix2x2
{
public:
 Matrix2x2(string filename)
 {
  ifstream f;
  f.open(filename);
  if (!f.good())
  {
   string s = "File Not found at: " + filename;
   throw exception(s.c_str());
  }
  int count = 0;
  string s;
  while (!f.eof())
  {
   getline(f, s);
   double x = atof(s.c_str());
   switch (count)
   {
    case 0: v[0][0] = x;break;
    case 1: v[0][1] = x;break;
    case 2: v[1][0] = x;break;
    case 3: v[1][1] = x;break;
   }

   count++;
  }
  f.close();
 }
```

Example 5

```
 void save(string filename)
 {
  ofstream f;
  f.open(filename);
  if (!f.good())
  {
   string s = "No rights to write to file: " + filename;
   throw exception(s.c_str());
  }
  int count = 0;
  double x=0;
  while (count<4)
  {
   switch (count)
   {
    case 0: x = v[0][0] ;break;
    case 1: x = v[0][1] ;break;
    case 2: x = v[1][0] ;break;
    case 3: x = v[1][1] ;break;
   }
   f << x << endl;
   count++;
  }
  f.close();
 }

private:
 double v[2][2];
};

int main(void) {
 Matrix2x2 m("matrix.txt");
 m.save("mcopy.txt");
 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. **Example 1**
        2. Example 2

Example 1

- Write a container class that holds the values of simple matrices (with a 2x2 size).
  - Add operators (i.e. << and >>) to print this matrix on the screen and get the values of this matrix.
  - Test this class with different values.
  - Use only a constructor to create a matrix and use operators to get the values from the user and to print the matrices to the user.

Example 1

```
#include <iostream>

using namespace std;
class Matrix2x2
{
public:
 Matrix2x2(double v1, double v2, double v3, double v4)
 {
  v[0][0] = v1;
  v[0][1] = v2;
  v[1][0] = v3;
  v[1][1] = v4;
 }

 void addValue(const double& rhs)
 {
  switch(lastAdded)
  {
  case 0:v[0][0] = rhs;break;
  case 1:v[0][1] = rhs;break;
  case 2:v[1][0] = rhs;break;
  case 3:v[1][1] = rhs;break;
  }
  lastAdded++;
  if (lastAdded > 4)
   lastAdded = 0;
 }
```

Example 1

```
 void operator<<(const double& rhs)
 {
  addValue(rhs);
 }
 void print(ostream &os) const
 {
  os << v[0][0] << " " << v[0][1] << endl;
  os << v[1][0] << " " << v[1][1] << endl;
 }
private:
 int lastAdded = 0;
 double v[2][2];
};
```

Example 1

```
istream& operator>>(istream& is, Matrix2x2& rhs)
{
 double value;
 is >> value;
 rhs.addValue(value);
 return is;
}
```

Example 1

```
ostream& operator<<(ostream& os, const Matrix2x2& rhs)
{
 rhs.print(os);
 return os;
}
```

Example 1

```
int main(void) {
 Matrix2x2 m(1, 5.5, 6, 9);
 //m << 3;//LEFT intentionally - for clarification
 //m << 4;
 //m << 5;
 //m << 6;
 cin >> m;
 cin >> m;
 cin >> m;
 cin >> m;
 cout << m;

 return 0;
}
```

Outline

1. Examples
    1. Inheritance
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    2. Exceptions
        1. Example 1
        2. Example 2
        3. Example 3
        4. Example 4
        5. Example 5
    3. Operators
        1. Example 1
        2. **Example 2**

Example 2

- Prepare a container class for a binary tree structure.
  - The binary tree is a tree data structure where each node has zero, one or two child nodes. These child nodes are referred to as the left child and the right child. This class consists of three fields:
    - a field with a value;
    - a pointer to the left child;
    - a pointer to the right child.
  - Implement a method to add a value to a tree, test it with some values (you can hard-code the values – testing should be easier and faster).
  - Overload the operator << to print all nodes inorder (inorder is a tree traversal method where you first traverse the left child inorder, then print the value of the current node and then traverse the right child inorder).

Example 2

```
#include <iostream>

using namespace std;
struct treeElement
{
 double value;
 treeElement *left;
 treeElement *right;
};

class BinaryTree
{
public:
 BinaryTree()
 {
  p = nullptr;
 }
 void insert(const double& rhs, treeElement *x) const
```

Example 2

```
void insert(const double& rhs, treeElement *x) const
{
 if (rhs< x->value)
 {
  if (x->left != nullptr)
   insert(rhs, x->left);
  else
  {
   x->left = new treeElement;
   x->left->value = rhs;
   x->left->left = nullptr;
   x->left->right = nullptr;
  }
 }
 else if (rhs >= x->value)
 {
  if (x->right != nullptr)
   insert(rhs, x->right);
  else
  {
   x->right = new treeElement;
   x->right->value = rhs;
   x->right->left = nullptr;
   x->right->right = nullptr;
  }
 }

}
```

Example 2

```
void insert(const double& rhs)
{
 if (p == nullptr)
 {
  p = new treeElement;
  p->value = rhs;
  p->left = nullptr;
  p->right = nullptr;
 }
 else
 {
  insert(rhs, p);
 }
}

void operator<<(const double& rhs)
{
 insert(rhs);
}
```

Example 2

```
 void operator<<(const double& rhs)
 {
  insert(rhs);
 }

 void printInorder(ostream &os, treeElement *x)
 {
  if (x->left != nullptr) printInorder(os, x->left);
  os << x->value << endl;
  if (x->right != nullptr) printInorder(os, x->right);
 }
 void print(ostream &os)
 {
  if (p!=nullptr)
   printInorder(os, p);
 }
private:
 treeElement *p;
};


ostream& operator<<(ostream& os, BinaryTree& rhs)
{
 rhs.print(os);
 return os;
}
```

Example 2

```
int main(void) {
 BinaryTree tree;
 double value;
 for (unsigned i = 0; i < 3; i++)
 {
  cin >> value;
  tree.insert(value);
 }
 //tree.insert(3);//LEFT intentionally - for clarification
 //tree.insert(5);
 //tree.insert(2);
 cout << tree;
 return 0;
}
```