/// Input Mismatch

#include <iostream>

#include <stdexcept>

using namespace std;

int main()

{

int num;

try

{

cout<<"Enter no ";

cin>>num;

if(cin.fail())

throw "Input must be integer";

cout<<"No is "<<num<<endl;

}

catch(const char \*msg)

{

cout<<msg<<endl;

}

cout<<"Hello class"<<endl;

cout<<"C++ exception"<<endl;

return 0;

}

……………………………………………….

/// Input Mismatch

#include <iostream>

#include <stdexcept>

using namespace std;

int main()

{

int num;

try

{

cout<<"Enter no ";

cin>>num;

if(cin.fail())

throw runtime\_error("input must be integer");

cout<<"No is "<<num<<endl;

}

catch(runtime\_error& e)

{

cout<<"Error..."<<e.what()<<endl;

}

cout<<"Hello class"<<endl;

cout<<"C++ exception"<<endl;

return 0;

}

………………………………………………..

//Input Mismatch + denominator must be > Zero

/// C++ Exceptions

#include <iostream>

#include<stdexcept>

using namespace std;

int main()

{

int no1, no2;

try

{

cout<<"Enter No-1 ";

cin>>no1;

if(cin.fail())

throw runtime\_error("No-1 must be integer");

cout<<"Enter No-2 ";

cin>>no2;

if(cin.fail())

throw runtime\_error("No-2 must be integer");

if(no2 == 0)

throw runtime\_error("/ by zero");

cout<<"Division result is "<<(float) no1/no2<<endl;

}

catch(runtime\_error& e)

{

cout<<"Error....."<<e.what()<<endl;

}

cout<<"Hello Class"<<endl;

cout<<"C++ Exceptions"<<endl;

return 0;

}

……………………………………………………….

/// stack Class with build in Exception

#include <iostream>

#include <stdlib.h>

#include <stdexcept>

using namespace std;

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

void Push(int var)

{

if(top >= 4)

{

cout<<"Stack overflow"<<endl;

exit(1);

}

arr[++top] = var;

}

int pop()

{

if(top == -1)

{

cout<<"Stack under flow"<<endl;

exit(1);

}

return arr[top--];

}

};

int main()

{

Stack s;

s.Push(11);

s.Push(12);

s.Push(13);

s.Push(14);

s.Push(15);

//s.Push(16);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

///cout<<"value is "<<s.pop()<<endl;

cout<<"Bye Bye"<<endl;

return 0;

}

…………………………………………………………………

#include <iostream>

#include <stdlib.h>

#include <stdexcept>

using namespace std;

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

void Push(int var)

{

if(top >= 4)

throw runtime\_error("Stack overflow");

arr[++top] = var;

}

int pop()

{

if(top == -1)

throw runtime\_error("Stack under flow");

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(12);

s.Push(13);

s.Push(14);

s.Push(15);

///s.Push(16);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

}

catch(runtime\_error& e)

{

cout<<"Error..."<<e.what()<<endl;

}

cout<<"Bye Bye"<<endl;

return 0;

}

……………………………………………………….

#include<iostream>

#include<stdexcept>

using namespace std;

class Distance

{

private:

int feets;

float inches;

public:

Distance(): feets(0),inches(0.0f){}

friend istream& operator>>(istream&, Distance&);

friend ostream& operator<<(ostream&,Distance&);

Distance& operator / (const Distance& d)

{

if(d.feets == 0 || d.inches == 0.0f)

throw "/ by zero (Denominator)";

feets = feets / d.feets;

inches = inches / d.inches;

return \*this;

}

};

istream& operator >>(istream& in, Distance& d)

{

cout<<"Enter Feets ";

in>>d.feets;

if(cin.fail())

throw runtime\_error("Feets must be Integer");

//throw "Feets must be Integer";

cout<<"Enter Inches ";

in>>d.inches;

if(cin.fail())

throw runtime\_error("Feets must be Integer ");

//throw "Inches must be Float";

}

ostream& operator << (ostream& out, Distance& d)

{

out<<"Distance is "<<d.feets<<'\''<<d.inches<<'\"'<<endl;

}

int main()

{

try

{

Distance d1, d2, d3;

cin >> d1;

cin >> d2;

d3 = d1 / d2;

cout<<"\nDivision result is "<<endl;

cout << d3;

}

catch(const char\* msg)

{

cout<<"Error."<<msg<<endl;

}

catch(runtime\_error& e)

{

cout<<"Error....."<<e.what()<<endl;

}

return 0;

}

……………………………………………..

/// stack Class with build in Exception (inherited)

#include <iostream>

#include <stdlib.h>

#include <stdexcept>

using namespace std;

class Stack : runtime\_error

{

private:

int arr[5];

int top;

public:

Stack(): runtime\_error(""), top(-1){}

void Push(int var)

{

if(top >= 4)

{

throw runtime\_error ("Stack overflow");

}

arr[++top] = var;

}

int pop()

{

if(top == -1)

{

throw runtime\_error ("Stack under flow");

}

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(12);

s.Push(13);

s.Push(14);

s.Push(15);

s.Push(16);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

///cout<<"value is "<<s.pop()<<endl;

}

catch(runtime\_error& e)

{

cout<<"Error..."<<e.what()<<endl;

}

cout<<"Bye Bye"<<endl;

return 0;

}

……………..

////User Define Exception Handling class

#include<iostream>

#include<stdexcept>

using namespace std;

class DivideByZero : public runtime\_error

{

public:

DivideByZero():runtime\_error(){}

DivideByZero(char ch[]):runtime\_error(ch){}

};

int main()

{

try

{

int num,dnum;

cout<<"Enter num ";

cin>>num;

cout<<"Enter D-num ";

cin>>dnum;

if(dnum == 0)

{

throw DivideByZero (“Error…/ by Zero”);

}

cout<<"Result :"<<num / dnum<<endl;

}

catch(DivideByZero d)

{

cout<<d.what()<<endl;

}

return 0;

}

………………………………………..

//user define exception Stack with separate Exception

#include <iostream>

#include <stdexcept>

using namespace std;

class MyException : public runtime\_error

{

public:

MyException(char ch[]):runtime\_error(ch){}

};

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

void Push(int var)

{

if(top >= 4)

throw MyException("Stack Over Flow");

arr[++top] = var;

}

int pop()

{

if(top == -1)

throw MyException("Stack Under Flow");

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

//s.Push(11);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

// cout<<"value is "<<s.pop()<<endl;

}

catch(MyException e)

{

cout<<e.what()<<endl;

}

return 0;

}

……………………………………….

//user define exception within class

#include <iostream>

#include <stdexcept>

using namespace std;

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

class Error//:public runtime\_error

{};

void Push(int var)

{

if(top >= 4)

throw Error();

arr[++top] = var;

}

int pop()

{

if(top == -1)

throw Error();

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

//s.Push(11);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

}

catch(Stack::Error)

{

cout<<"Error...Stack full/empty"<<endl;

}

cout<<"BSCS regular"<<endl;

return 0;

}

/////////////////’

//user define exception within class (User Define inherit with runtime\_error class)

#include <iostream>

#include <stdexcept>

using namespace std;

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

class Error:public runtime\_error

{

public:

Error(string s):runtime\_error(s){}

};

void Push(int var)

{

if(top >= 4)

throw Error("Error...Stack full");

arr[++top] = var;

}

int pop()

{

if(top == -1)

throw Error("Error...Stack empty");

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

//s.Push(11);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

}

catch(Stack::Error e)

{

cout<<e.what()<<endl;

}

cout<<"BSCS regular"<<endl;

return 0;

}

……………….

// //user define exception with arguments within class (User Define inherit with runtime\_error class)

#include <iostream>

#include <stdexcept>

using namespace std;

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

class Error:public runtime\_error

{

public:

int ptop;

Error(string s, int pt):runtime\_error(s), ptop(pt){}

};

void Push(int var)

{

if(top >= 4)

throw Error("Error...Stack full", top);

arr[++top] = var;

}

int pop()

{

if(top == -1)

throw Error("Error...Stack empty", top);

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

}

catch(Stack::Error e)

{

cout<<e.what()<<endl;

cout<<"Top value is "<<e.ptop<<endl;

}

cout<<"\nBSCS regular"<<endl;

return 0;

}

// multi exception with multi catch blocks

#include <iostream>

#include <stdexcept>

using namespace std;

class Stack

{

private:

int arr[5];

int top;

public:

Stack():top(-1){}

class Full:public runtime\_error

{

public:

Full(string s):runtime\_error(s){}

};

class Empty:public runtime\_error

{

public:

Empty(string s):runtime\_error(s){}

};

void Push(int var)

{

if(top >= 4)

throw Full("Error...Stack full");

arr[++top] = var;

}

int pop()

{

if(top == -1)

throw Empty("Error...Stack empty");

return arr[top--];

}

};

int main()

{

Stack s;

try

{

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

s.Push(11);

//s.Push(11);

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

cout<<"value is "<<s.pop()<<endl;

}

catch(Stack::Full e)

{

cout<<e.what()<<endl;

}

catch(Stack::Empty e)

{

cout<<e.what()<<endl;

}

cout<<"\nBSCS regular"<<endl;

return 0;

}

//

For Home Work

Question: Create a Professor class that has data members to holds the *Id* (int), *name* (string) and *Pub* (int). Class also includes parameterized *constructors* and overloaded *insertion* (<<) and *extraction* *(<<)* operators’ that displays and get all fields of class Professor. Create an ProException class that holds *EstimPub* (type int). When the user enters Professor data, if the pub is below then 10, then throw an ProException object with an appropriate message *(Pass this String to the ProException’s parent so it can be used in a what () call)*. Write a main () function that instantiates a Professor object, allows the user to enter data, and displays the data members.

1st method

//user define exception

#include <iostream>

#include <stdexcept>

#include <cstring>

using namespace std;

class Professor

{

private:

int id, pub;

char name[20];

public:

class ProException :public runtime\_error

{

public:

int expub;

ProException(int p, string s): runtime\_error(s), expub(p){}

};

Professor(int i, char na[], int p) :id(i)

{

strcpy(name, na);

if(p <10)

{

throw ProException(i, "Publication must be >= 10");

}

}

friend void operator >> (istream&, Professor&);

friend void operator << (ostream&, Professor&);

};

void operator >> (istream& s, Professor& p)

{

cout<<"Enter id ";

s>>p.id;

s.ignore();

cout<<"Enter Name ";

s.get(p.name,20);

cout<<"Enter Publication ";

s>>p.pub;

if(p.pub <10)

throw Professor::ProException(p.id, "Publication must be >= 10");

}

void operator << (ostream& s, Professor& p)

{

s<<endl;

s<<"Professor Id "<<p.id<<endl;

s<<"Professor Name "<<p.name<<endl;

s<<"Professor Publication "<<p.pub<<endl;

}

int main()

{

try

{

Professor p(1,"",100);

cin>>p;

cout<<p;

}

catch(Professor::ProException& e)

{

cout<<"Error.."<<" Professor id is "<<e.expub<<endl;

cout<<e.what();

}

return 0;

}

2nd method

//user define exception

#include <iostream>

#include <stdexcept>

#include <cstring>

using namespace std;

class ProException :public runtime\_error

{

public:

int expub;

ProException(int p, char s[]): runtime\_error(s), expub(p){}

};

class Professor

{

private:

int id, pub;

char name[20];

public:

Professor(int i, char na[], int p) :id(i)

{

strcpy(name, na);

if(p <10)

{

throw ProException(i, "Publication must be >= 10");

}

}

friend void operator >> (istream&, Professor&);

friend void operator << (ostream&, Professor&);

};

void operator >> (istream& s, Professor& p)

{

cout<<"Enter id ";

s>>p.id;

s.ignore();

cout<<"Enter Name ";

s.get(p.name,20);

cout<<"Enter Publication ";

s>>p.pub;

if(p.pub <10)

throw ProException(p.id, "Publication must be >= 10");

}

void operator << (ostream& s, Professor& p)

{

s<<endl;

s<<"Professor Id "<<p.id<<endl;

s<<"Professor Name "<<p.name<<endl;

s<<"Professor Publication "<<p.pub<<endl;

}

int main()

{

try

{

Professor p(1,"",100);

cin>>p;

cout<<p;

}

catch(ProException& e)

{

cout<<endl;

cout<<"Error.."<<" Professor id is "<<e.expub<<endl;

cout<<e.what();

cout<<endl;

}

return 0;

}