```
// assert1.cpp

// to illustrate using the assert library function using NDEBUG
//
        #include <iostream>
        #include <cassert>
        using namespace std;
        int main()
        {
         int myInt;
    cout << "Please enter an integer greater than zero" << endl;
         cin >> myInt ;
         assert (myInt > 0) ;
         cout << "The value of myInt is "  << myInt << endl;
        }
```

```
Please enter an integer greater than zero
-4
Assertion failed: myInt > 0, file ..\main.cpp, line 15

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

```
// assert2.cpp
// to illustrate using the assert library function using NDEBUG
//
        #include <iostream>
         #define NDEBUG // must be before the #include <cassert>
        #include <cassert>

        using namespace std;

        int main()
        {
         int myInt;
    cout << "Please enter an integer greater than zero" << endl;
         cin >> myInt ;
         assert (myInt > 0) ;
         cout << "The value of myInt is "  << myInt << endl;
        }
Please enter an integer greater than zero
-3
The value of myInt is

// except1.cpp
//
#include <iostream>
using namespace std;

int main()
{
        int donuts;
        int milk;
        double dpg;

        try
        {
                cout << "Enter number of donuts:\n";
                cin >> donuts;
                cout << "Enter number of glasses of milk:\n";
                cin >> milk;

                if (milk <= 0)
                        throw donuts;

                dpg = donuts/double(milk);
                cout << donuts << " donuts.\n"
                         << milk << " glasses of milk.\n"
                         << "You have " <<dpg
                         << " donuts for each glass of milk.\n";
        }
        catch(int e)
        {
                cout << e << " donuts, and No          Milk!\n"
                         << "Go buy some milk.\n";
        }
        cout << "End of program.\n";
        return 0;
}
output
Enter number of donuts:
10
Enter number of glasses of milk:
-3
10 donuts, and No Milk!
Go buy some milk.
End of program.

// except2.cpp
// demonstrates exceptions using exception classes

#include <iostream>
using namespace std;
const int MAX = 3;


class Range {};
class Stack
  {
        public:
                Stack();                  // constructor
                void push(int var);       // put number on stack
                int pop();                // take number off stack
        private:
                int st[MAX];              // stack: array of any type
                static int top;                // number of top of stack
        };

 int Stack::top;


                        Stack::Stack()                  // constructor
                        {
                             top = -1;
                        }


                        void Stack::push(int var)       // put number on stack
                        {
                         if(top >= MAX-1)       // if stack full,
                        throw Range();       // throw exception
                        st[++top] = var;       // put number on stack
                        }

                        int Stack::pop()               // take number off stack
                        {
                          if(top < 0)           // if stack empty,
                         throw Range();      // throw exception
                        return st[top--];      // take number off stack
                        }

        int main()
        {
        Stack  myStack;      // myStack is object of class Stack<int>

     try
       {
       myStack.push(11);
       myStack.push(22);
       myStack.push(33);
       myStack.push(44);                         // oops: stack full

       cout << "1: " << myStack.pop() << endl;
       cout << "2: " << myStack.pop() << endl;
       cout << "3: " << myStack.pop() << endl;
       cout << "4: " << myStack.pop() << endl;  // oops: stack empty
       }


        catch(const Range &)     // exception handler
       {
```

```
    cout << "Stack Full or Empty" << endl;
    }
              return 0;
       }
Output
Stack Full or Empty

// except3.cpp
// demonstrates two exception handlers with the exception classes defined
// within a class

#include <iostream>
using namespace std;
const int MAX = 3;

class Stack
       {

       public:

              Stack();                 // constructor
              void push(int var);      // put number on stack
              int pop();               // take number off stack
              class Full { };          // exception class
              class Empty { };         // exception class

       private:
              int st[MAX];             // stack: array of any type
              static int top;          // number of top of stack
       };

       int Stack::top;
```

```
        Stack::Stack()              // constructor
                    {
                        top = -1;
                    }

                void Stack::push(int var)     // put number on stack
                    {
                     if(top >= MAX-1)      // if stack full,
                    throw Full();      // throw Full exception
                    st[++top] = var;      // put number on stack
                    }

                    int Stack::pop()           // take number off stack
                    {
                      if(top < 0)          // if stack empty,
                        throw Empty();     // throw Empty exception
                    return st[top--];     // take number off stack
                    }

int main()
  {
  Stack myStack;

  try
    {
    myStack.push(11);
    myStack.push(22);
    myStack.push(33);
    myStack.push(44);
    myStack.push(55);
        }

        catch(const Stack::Full &)
    {
    cout << "Stack Full" << endl;
    }
```

```
   try
     {
     cout << "1: " << myStack.pop() << endl;
     cout << "2: " << myStack.pop() << endl;
     cout << "3: " << myStack.pop() << endl;
     cout << "4: " << myStack.pop() << endl;  // oops: stack empty
     }

   catch const Stack::Empty &))
     {
     cout << "Stack Empty" << endl;
      }

    cout << "The end" << endl;
          return 0;
   }
```

```
Stack Full
1: 33
2: 22
3: 11
Stack Empty
The end
```

```
// except4.cpp
// demonstrates two exception handlers
// consecutive catch blocks

#include <iostream>
const int MAX = 3;

class Stack
          {

          public:
```

```
          Stack();                   // constructor
          void push(int var);       // put number on stack
          int pop();               // take number off stack
          class Full { };          // exception class
          class Empty { };         // exception class

     private:
          int st[MAX];             // stack: array of any type
          static int top;                // number of top of stack
     };

int Stack::top;


     Stack::Stack()                  // constructor
               {
                    top = -1;
               }

          void Stack::push(int var)       // put number on stack
               {
                if(top >= MAX-1)       // if stack full,
                    throw Full();       // throw Full exception
               st[++top] = var;       // put number on stack
               }

               int Stack::pop()                // take number off stack
               {
                 if(top < 0)           // if stack empty,
                    throw Empty();      // throw Empty exception
               return st[top--];      // take number off stack
               }

               int main()
                 {
                 Stack myStack;
```

```
try
  {
  myStack.push(11);
  myStack.push(22);
  myStack.push(33);
  myStack.push(44);
 myStack.push(55);

  cout << "1: " << myStack.pop() << endl;
  cout << "2: " << myStack.pop() << endl;
  cout << "3: " << myStack.pop() << endl;
  cout << "4: " << myStack.pop() << endl;  // oops: stack empty
  }

 catch(const Stack::Empty &)
  {
  cout << "Stack Empty" << endl;
              }

       catch(const Stack::Full &)
              {
              cout << "Stack Full" << endl;
              }

     cout << "The end" << endl;
     return 0;
}
```

Output

```
Stack Full
The end
```

```
// except5.cpp
// exceptions with Distance class
#include <iostream>
#include <string>
using namespace std;
```

```
class Distance                    // English Distance class
  {

  public:
              class InchesExceeded { };        // exception class

              Distance() ;                // constructor (no args)
              Distance(int ft, float in);  // constructor (two args)
              void getdist()  ;           // get length from user
              void showdist() ;            // display distance
       private:
    int feet;
    float inches;
        };

                     Distance::Distance()                 // constructor (no args)
                     {
                     feet = 0; inches = 0.0;
                     }

                     Distance::Distance(int ft, float in)  // constructor (two args)
                     {
                     if(in >= 12.0)           // if inches too big,
                            throw InchesExceeded();     // throw exception
                     feet = ft;
                     inches = in;
                     }

              void   Distance::getdist()             // get length from user
                     {
                     cout << "\nEnter feet: ";  cin >> feet;
                     cout << "Enter inches: ";  cin >> inches;
                     if(inches >= 12.0)       // if inches too big,
                            throw InchesExceeded();     // throw exception
                     }

              void   Distance::showdist()            // display distance
                     {
```

```
                    cout << feet << "\'-" << inches << '\"';
                    }

int main()
      {
      try
            {
            Distance dist1(17, 3.5);    // 2-arg constructor
            cout << "\ndist1 = ";
            dist1.showdist();
            Distance dist2;             // no-arg constructor
            cout << "\ndist2 = ";
            dist2.showdist();
            dist2.getdist();            // get distance
            cout << "\ndist2 = ";
            dist2.showdist();
            }

      catch(const Distance::InchesExceeded &)     // catch exceptions
            {
            cout << "\nInitialization error: "
                          "inches value is too large.";
            }
      }
```

output

```
dist1 = 17'-3.5"
dist2 = 0'-0"
Enter feet: 30
Enter inches: 66

Initialization error: inches value is too large.
```

```
// except6.cpp
// exceptions with Distance class
#include <iostream>
#include <string>           // for strcpy()
using namespace std;
```

```
class InchesExceeded              // exception class
     {
       public:
       InchesExceeded();
                 InchesExceeded(float in);
                 private:
                 float iValue;       // for faulty inches value
          };

class Distance                    // English Distance class
   {


         public:
                 Distance() ;                 // constructor (no args)
                 Distance(int ft, float in);  // constructor (two args)
                 void getdist()  ;            // get length from user
                 void showdist() ;             // display distance
          private:
     int feet;
     float inches;
           };

                 Distance::Distance()                 // constructor (no args)
                 {
                         feet = 0; inches = 0.0;
                 }

                 Distance::Distance(int ft, float in)  // constructor (two args)
                 {
                         if(in >= 12.0)          // if inches too big,
                         throw InchesExceeded (in);
                         feet = ft;
                         inches = in;
                 }

                 void    Distance::getdist()             // get length from user
```

```
                  {
                  cout << "\nEnter feet: ";  cin >> feet;
                  cout << "Enter inches: ";  cin >> inches;
                  if(inches >= 12.0)       // if inches too big,
                          throw InchesExceeded( inches);
                  }

          void    Distance::showdist()            // display distance
                  {
                  cout << feet << "\'-" << inches << '\"';
                  }

          InchesExceeded::InchesExceeded(float in)   // 1-arg constructor
           {
                   iValue = in;               // store inches
                  cout   << ".\n   Inches value of " << iValue
                    << " is too large." << endl;
                                       };
int main()
  {
       try
          {
          Distance dist1(17, 333);    // 2-arg constructor
          cout << "\ndist1 = ";
          dist1.showdist();
          }

       catch(InchesExceeded &inches) // exception handler
          {
          cout << " 2 argument catch " << endl;
          }

    try
          {
          Distance dist1(17, 3.5);    // 2-arg constructor
          cout << "\ndist1 = ";
          dist1.showdist();
          Distance dist2;            // no-arg constructor
```

```
                    cout << "\ndist2 = ";
                    dist2.showdist();
                    dist2.getdist();            // get distance
                    cout << "\ndist2 = ";
                    dist2.showdist();

                    }
                    catch(InchesExceeded &inches)// exception handler
                    {
                    cout << " Get distance catch" << endl;
                    }
          }
output

   Inches value of 333 is too large.
 2 argument catch

dist1 = 17'-3.5"
dist2 = 0'-0"
Enter feet: 55
Enter inches: 777
.
   Inches value of 777 is too large.
 Get distance catch

// except7.cpp
// ************ missing a catch block************

#include <iostream>
using namespace std;

const int MAX = 3;

class Stack
          {

          public:
                    Stack();                    // constructor
                    void push(int var);       // put number on stack
                    int pop();                // take number off stack
                    class Full { };           // exception class
                    class Empty { };          // exception class

          private:
                    int st[MAX];              // stack: array of any type
                    static int top;                 // number of top of stack
          };

    int Stack::top;


          Stack::Stack()                  // constructor
                  {
                          top = -1;
                  }

                    void Stack::push(int var)       // put number on stack
                  {
                   if(top >= MAX-1)       // if stack full,
                          throw Full();       // throw Full exception
                  st[++top] = var;       // put number on stack
                  }

                  int Stack::pop()                // take number off stack
                  {
                    if(top < 0)           // if stack empty,
                          throw Empty();      // throw Empty exception
                  return st[top--];      // take number off stack
                  }

int main()
   {
   Stack myStack;

   try
```

```
    {
    myStack.push(11);
    myStack.push(22);
    myStack.push(33);
       myStack.push(44);
       myStack.push(55);

       cout << "1: " << myStack.pop() << endl;
    cout << "2: " << myStack.pop() << endl;
    cout << "3: " << myStack.pop() << endl;
    cout << "4: " << myStack.pop() << endl;  // oops: stack empty
    }

  catch(Stack::Empty &)
    {
    cout << "Stack Empty" << endl;
              }

       cout << "The end" << endl;
  }
```

output

```
This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

```
// except8.cpp
// ************Catch all************
// consecutive catch blocks

#include <iostream>
using namespace std;

const int MAX = 3;

class Stack
       {

       public:
            Stack();                  // constructor
            void push(int var);       // put number on stack
            int pop();                // take number off stack
            class Full { };           // exception class
            class Empty { };          // exception class

        private:
            int st[MAX];              // stack: array of any type
            static int top;                 // number of top of stack
        };

    int Stack::top;


        Stack::Stack()                // constructor
                {
                    top = -1;
                }

            void Stack::push(int var)       // put number on stack
                {
                 if(top >= MAX-1)       // if stack full,
                    throw Full();       // throw Full exception
                st[++top] = var;        // put number on stack
                }

                int Stack::pop()               // take number off stack
                {
                  if(top < 0)            // if stack empty,
                    throw Empty();      // throw Empty exception
                return st[top--];       // take number off stack
                }

int main()
  {
  Stack myStack;
```

```
   try
     {
     myStack.push(11);
     myStack.push(22);
     myStack.push(33);
        myStack.push(44);
        myStack.push(55);

        cout << "1: " << myStack.pop() << endl;
     cout << "2: " << myStack.pop() << endl;
     cout << "3: " << myStack.pop() << endl;
     cout << "4: " << myStack.pop() << endl;  // oops: stack empty
     }

   catch(Stack::Empty &)
     {
     cout << "Stack Empty" << endl;
               }
   catch(...)
       {
       cout << "Catch all" << endl;
               }

        cout << "The end" << endl;
   }
```

```
Output
Catch all
The end
```

```
// except9.cpp
// demonstrates ***** execution after an exception*********
// consecutive catch blocks

#include <iostream>
using namespace std;

const int MAX = 3;
```

```
class Stack
        {

        public:

                Stack();                    // constructor
                void push(int var);       // put number on stack
                int pop();                // take number off stack
                class Full { };           // exception class
                class Empty { };          // exception class

        private:
                int st[MAX];              // stack: array of any type
                static int top;                  // number of top of stack
        };

 int Stack::top;


        Stack::Stack()                  // constructor
                        {
                                top = -1;
                        }

                void Stack::push(int var)       // put number on stack
                        {
                         if(top >= MAX-1)       // if stack full,
                                throw Full();       // throw Full exception
                        st[++top] = var;       // put number on stack
                        }

                        int Stack::pop()                // take number off stack
                        {
                          if(top < 0)            // if stack empty,
                                throw Empty();      // throw Empty exception
                        return st[top--];      // take number off stack
                        }
```

```
void playWithStack(int topOfStack);
int main()
  {
       playWithStack(MAX+5);
       return 0;
      }

      void playWithStack(int topOfStack)
      {
  try
             {
             Stack myStack;
             int index;
             for (index=0; index<topOfStack; index++)
             {
                    myStack.push(index);
             }


             cout << "1: " << myStack.pop() << endl;
      cout << "2: " << myStack.pop() << endl;
      cout << "3: " << myStack.pop() << endl;
             cout << "4: " << myStack.pop() << endl;  // oops: stack empty
    }

  catch(Stack::Empty &)
    {
    cout << "Stack Empty" << endl;
             }

       catch(Stack::Full &)
             {
             cout << "Stack Full - Lets try again" << endl;
             playWithStack(MAX);
             }

  }
```

Output

```
Stack Full - Lets try again
1: 2
2: 1
3: 0
Stack Empty
```

```
// except10.cpp
// out_of_range example
#include <iostream>       // std::cerr
#include <stdexcept>      // std::out_of_range
#include <vector>
using namespace std;// std::vector

int main () {
  std::vector<int> myvector(10);
  try {
    myvector.at(20)=100;      // vector::at throws an out-of-range
  }
  catch (const std::out_of_range& oor) {
    std::cerr << "Out of Range error: " << oor.what() << '\n';
  }
  return 0;
}
```

Output

```
Out of Range error: vector::_M_range_check: __n (which is 20) >= this-
>size() (which is 10)
```

```
// except11.cpp
#include <iostream>
using namespace std;
int main()
{

        int * list[100];
        try
        {
                for (int index = 0; index < 100; index++)
                {
                list[index] = new int[50000000];
                cout << "created list[" << index <<"] of 50000000 components\n";
                }
        }

        catch(bad_alloc &message)
        {
                cout << "In bad_alloc catch block "
                        << message.what() << endl;
        }

        cout << "End of program.\n";
        return 0;
}
```

Output

```
created list[0] of 50000000 components
created list[1] of 50000000 components
created list[2] of 50000000 components
created list[3] of 50000000 components
created list[4] of 50000000 components
created list[5] of 50000000 components
created list[6] of 50000000 components
created list[7] of 50000000 components
created list[8] of 50000000 components
In bad_alloc catch block std::bad_alloc
End of program.
```

```
// except12.cpp
#include <iostream>
using namespace std;
int main()
{

        int * list[100];

        {
                for (int index = 0; index < 100; index++)
```

```
            {
            list[index] = new int[50000000];
            cout << "created list[" << index <<"] of 50000000 components\n";
            }
        }


        cout << "End of program.\n";
        return 0;
}
```

```
created list[0] of 50000000 components
created list[1] of 50000000 components
created list[2] of 50000000 components
created list[3] of 50000000 components
created list[4] of 50000000 components
created list[5] of 50000000 components
created list[6] of 50000000 components
created list[7] of 50000000 components
created list[8] of 50000000 components
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
```

```
// except13.cpp

// ***Handle division by zero, division by a negative integer,
// and input failure exceptions***

#include <iostream>
#include <string>
using namespace std;


using namespace std;

int main()
{
    int dividend, divisor = 1, quotient;             //Line 1

    string inpStr
       = "The input stream is in the fail state.";  //Line 2

    try                                              //Line 3
    {
        cout << "Line 4: Enter the dividend: ";      //Line 4
        cin >> dividend;                             //Line 5
        cout << endl;                                //Line 6
        cout << "Line 7: Enter the divisor: ";      //Line 7
        cin >> divisor;                             //Line 8
        cout << endl;                               //Line 9

        if (divisor == 0)                           //Line 10
            throw divisor;                          //Line 11
        else if (divisor < 0)                       //Line 12
            throw string("Negative divisor.");      //Line 13
        else if (!cin)                              //Line 14
            throw inpStr;                           //Line 15

        quotient = dividend / divisor;              //Line 16

        cout << "Line 17: Quotient = " << quotient
             << endl;                               //Line 17
    }
    catch (int x)                                   //Line 18
    {
        cout << "Line 19: Division by " << x
             << endl;                               //Line 19
    }
    catch (string &myString)                        //Line 20
    {
        cout << "Line 21: " << myString << endl;           //Line 21
    }

    return 0;                                       //Line 22
}
```

Output:

Execution 1

```
Line 4: Enter the dividend: 7

Line 7: Enter the divisor: 0

Line 19: Division by 0
```

Execution 2

```
Line 4: Enter the dividend: 35

Line 7: Enter the divisor: 7

Line 17: Quotient = 5
```

Execution 3

```
Line 4: Enter the dividend: 5

Line 7: Enter the divisor: -1

Line 21: Negative divisor
```

Execution 4

```
Line 4: Enter the dividend: uuu

Line 7: Enter the divisor:
```

```
Line 21: The input stream is in the fail state.
```

```
// except14.cpp
#include <iostream>
#include <stdexcept>
#include <string>
using namespace std;

int main()
{
   string sentence;                          //Line 1
   string str1, str2, str3;                   //Line 2

   try                                   //Line 3
   {
      sentence = "Testing string exceptions!";    //Line 4
      cout << "Line 5: sentence = " << sentence
          << endl;                          //Line 5
      cout << "Line 6: sentence.length() = "
          << static_cast<int>(sentence.length())
          << endl;                          //Line 6

      str1 = sentence.substr(8, 20);              //Line 7
      cout << "Line 8: str1 = " << str1
          << endl;                          //Line 8

      str2 = sentence.substr(28, 10);              //Line 9
      cout << "Line 10: str2 = " << str2
          << endl;                          //Line 10

      str3 = "Exception handling. " + sentence;   //Line 11
      cout << "Line 12: str3 = " << str3
          << endl;                          //Line 12

   }

   catch (length_error &le)                      //Line 15
```

```
    {
        cout << "Line 16: In the length_error "
             << "catch block: " << le.what()
             << endl;                        //Line 16
    }

    catch (out_of_range &re)                     //Line 13
    {
        cout << "Line 14: In the out_of_range "
             << "catch block: " << re.what()
             << endl;                        //Line 14
    }

    return 0;                                //Line 17
}
```

Output

```
Line 5: sentence = Testing string exceptions!
Line 6: sentence.length() = 26
Line 8: str1 = string exceptions!
Line 14: In the out_of_range catch block: basic_string::substr: __pos (which is 28)
> this->size() (which is 26)
```