```
// pop_back() removes the last element of the vector
   myvec.pop_back(); //15
   myvec.pop_back(); //12
   cout<<myvec.size(); //0
// Using vectors as parameters
int total (Vector<int> v)
{  int sum=0;
   for(int i=0; i<v.size(); i++)
            sum+=v[i];
   return sum;
} //end func.
```

Test #2, Ch. 17, 18 (Linked List, stack, and Queue), Sat. 4-27-24.

Multiple Choice + Written (4 Questions)

Week 12, Sat:

Exception Handling: C++ provides a built in Handling the errors that is called "Exception Handling".

e.g. Dividing by Zero, Square root of negative number, to access a file in a folder and that file does not exist, ..., our own exceptions (e.g. width of a rectangle cannot be negative, ...).

In C++ Exception Handling is built upon 3 Keywords: try, catch, throw.

If an exception (error) occurs within the try block, it is thrown (using throw), then it will be caught by a catch block and we can prompt the user with an error message and rest of the program will be executed.

Serious Error: e.g. Syntax or Grammer Error, we cannot use try and catch, we must fix them (Compile error), Checked errors.

Less Serious Error: We can use catch and try, to give a message to the user about the error and execution of program will not be stopped by Compiler.

```
try
{ . . .
    // Codes
}
Catch (type1 arg1)
{
    // Catch block
    . . .
}
Catch (type2 arg2)
{ . . .
}
. . .
Catch (tyen argn)
{ . . .
}
// execution Continues
. . .
```

No Code here

No Code here

The try block must Contain the portion of the program that we want to monitor for errors. When an exception is thrown, it is caught by Corresponding Catch block. Then it processes the exception and Continues the execution of the rest of the program. If there is no proper Catch, program will be stopped the Compiler. If the data type specified by a catch, matches that of the exception that Catch statement is executed and other catch blocks will be ignored and it is not going back to the try block but it will continue with the rest of the code after all catch blocks. The general form of a throw statement is:

throw exception; // is the value thrown (Caused the error)

throw must be executed either from within the try block or from any func. the Code within the try block Calls (directly or indirectly)

Example 1:

```
int main()
{   cout << "Start\n";

    try{  cout << "Inside try block\n";
          throw 10; // throw an error
          cout << "This line will not be executed." << endl;
    }
    catch (int i)
    {  cout << "Caught One! Number is: " << i << endl;
    }
    cout << "Continue the rest.";
    :      :
    return 0;
} // end main
// output
```

```
Start
Inside try block
Caught One! Number is: 10
Continue the rest
```

Exceptions: indicate that something unexpected has occurred or been detected while program is running. (run time error).
Exceptions allow program to deal with the problem in a controlled manner.
It can be as simple or complex as program design requires.

Terminologies:

- **Exception:** Object or value that signals an error
- **Throw an exception:** Sends a signal that an error has occurred.
- **Catch / handles an exception:** processes the exception and interprets the signal.
- **throw:** followed by an argument, is used to throw an exception.
- **try:** followed by a block { }, is used to invoke code that throws an exception.
- **catch:** " " " " " " " to detect and process the exception thrown in preceding try block. Takes a parameter the "data type" thrown.

Example 2: // func. that throws an exception.

```
int totalDays (int days, int week)
{ if (days<0) || (days>7) //programmer's defined exception
        throw "Invalid number of days!";
  else
        return (7*weeks + days);
}
...
int main()
{ ...
  try { totDays = totalDays (days, weeks); //func. call
        cout << "Total days: " << totDays << endl;
      }
  catch (char *msg)
  { cout << "Error: " << msg << endl;
  }
  // continues with the rest of the code
  ...
```

- An exception will not be caught if:
  - It is thrown from outside of a try block.
  - There is no catch block matches the data type of the thrown exception.
  - If an exception is not caught, the program will terminate by compiler.

In OOP: Exceptions and Objects

- An exception class can be defined in a class and thrown as an exception by a member function.
- An exception class may have:
  - No members: Used only to signal an error
  - members: pass error data to catch block.

A class can have more than one exception class.

Example 3:

```
Class Rectangle
{ private: double width;
           double length;
  public: // Exception class
          class NegativeSize
          { }; // empty class
  Rectange() // default constructor
  { width=0.0; length=0.0; }
  void setWidth(double); // prototype
   "   setLength(double); // "
   ...  // getWidth, getLength
}//end class
void Rectangle:: setWidth(double w)
{ if (w>0)
      width=w;
  else throw NegativeSize(); // an object of NegativeSize class
// The throw statement's argument NegativeSize(), Causes an instance of NegativeSize
// Class to be Created and thrown an exception
}//end setWidth
void Rectangle:: setLength(double len)
{ if (len>0)
      length=len;
  else throw NegativeSize();
}//end setLength
...
int main()
{ double wid, leng;
  // We get wid and leng from user using cout and cin
  Rectangle rect1;
```

```
try { rect1.setWidth(wid);
      rect1.setLength(leng);
      cout<<"Area is: "<<rect1.getArea();
    }
catch( Rectangle::NegativeSize)
{  cout <<"Error: A negative value was entered.\n";
}
cout<<"Program Continues.\n";
return 0;
}//end main
```

This is data type

Multiple Exceptions:

```
class NegativeWidth
{ };
class NegativeLength
{ };
// in setWidth func.
  throw NegativeWidth();
// in setLength func.
  throw NegativeLength();
```

In C++, `catch(...)` → Catches all exceptions

Example #4:

```
#include <iostream>
...
void func (int num)
{  try{  if (num==0) throw num; // throw int
         if (num==1) throw 'a'; //  "  char
         if (num==2) throw 74.15; //  "  double
      }
      catch (...)
      { //Catch all exceptions
        cout << "Caught One!" <<endl;
      } //end
}//end func.
```

```
int main()
{   cout << "start\n";
    func(0);
    func(1);
    func(2);
    cout << "end \n";
    return 0;
}//end main
```

Output: start
Caught One!
Caught One!
Caught One!
end

When we have an exception in a try block, it goes to the specific catch block and never goes to the try block, it continues with the rest of the code.

Example #5: File I/O exception:

```
try {   ifstream file("mydata.txt");
        if(!file.is_open())
                throw "File not found.";
        else
            // File operations here
            file.close();
    }
catch(const char *err)
{   cout << "Exception Caught: " << err << endl;
}
```

Example #6: Division By Zero Exception:

```
try {   int a=10;
        int b=0;
        if(b==0)
                throw "Division by Zero";
        else int result=a/b;
        cout << "Result is: " << result << endl;
    }
catch(const char *err)
{   cout << "Exception Caught: " << err << endl;}
```

Example #7: Invalid argument exception with stoi

```
try {   string str= "123abc";
        int value = stoi (str); // Converting invalid string to integer
    }
Catch ( const invalid_argument & err )
{  cout << " Exception Caught : " << err.what ()<<endl;
```