|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Description | Change code | Expected Output | Passed |
| 1 | Check constructor initialises object | Vector<int> testVec;  printIntVec(testVec); | Length = 0  Size = 0  NULL | Y |
| 2 | Check SetSize function | Vector<int> testVec;  testVec.SetSize(10);  printIntVec(testVec); | Length = 0  Size = 10  NOT NULL | Y |
| 3 | Check SetSize function with 0 | Vector<int> testVec;  testVec.SetSize(0); | Zero or less!!!  Length = 0  Size = 0  NULL | Y |
| 4 | Check copy function | Vector<int> testVec, testVec02, testVec.SetSize;  testVec.SetArray(testArr, 3);  if(testVec02.CopyVec(testVec))  cout << "Vector copied" << endl; printIntVec(testVec); | Vector copied  Length = 3  Size = 3  NOT NULL  Element 0 is: 12  Element 1 is: 23  Element 2 is: 45 | Y |
| 5 | Check contructor with size parameter | Vector<int> testVec(11);  printIntVec(testVec); | Length = 0  Size = 11  NOT NULL | Y |
| 6 | Check copy contructor | Vector<int> testVec;  testVec.SetArray(testArr, 3);  Vector<int> testVec02(testVec);  printIntVec(testVec); | Length = 3  Size = 3  NOT NULL  Element 0 is: 12  Element 1 is: 23  Element 2 is: 45 | Y |
| 7 | Check overloaded [] const function.  Cassert tested but left out of test. | Vector<int> testVec01;  testVec01.SetArray(testArr, 3);  const Vector<int> testVec02(testVec01);  cout << testVec02[1] << endl;  printIntVec(testVec02); | 23  Length = 3  Size = 3  NOT NULL  Element 0 is: 12  Element 1 is: 23  Element 2 is: 45 | Y |
| 8 | Test overloaded operater [] | Vector<int> testVec(3);  for(int i = 0; i < 3; i++)  testVec.AddItem(i);  for(int i = 0; i < 3; i++)  testVec[i] = i + 10;  printIntVec(testVec); | Length = 3  Size = 3  NOT NULL  Element 0 is: 10  Element 1 is: 11  Element 2 is: 12 | Y |
| 9 | Test GetArray | Vector<int> testVec;  int \*testArr;  testVec.SetArray(intArr, 3);  testVec.GetArray(testArr);  for(int i = 0; i < 3; i++)  cout << testArr[i] << endl; | 12  23  45 | Y |
| 10 | Test SetArray | Vector<string> testVec(3);  string strArr[] = {"Apples", "Oranges", "Pears"};  if(testVec.SetArray(strArr, 3))  printIntVec(testVec); | Length = 3  Size = 3  NOT NULL  Element 0 is: Apples  Element 1 is: Oranges  Element 2 is: Pears | Y |
| 11 | Test GetLength and GetSize | Vector<string> testVec;  testVec.SetArray(strArr, 3);  cout << testVec.GetLength() << endl;  cout << testVec.GetSize() << endl; | 3  3 | Y |
| 12 | Test overloaded = operator | Vector<string> testVec01, testVec02;  testVec01.SetArray(strArr, 3);  testVec02 = testVec01;  printIntVec(testVec02);} | Length = 3  Size = 3  NOT NULL  Element 0 is: Apples  Element 1 is: Oranges  Element 2 is: Pears | Y |
| 13 | Test Resize | Vector<string> testVec;  testVec.SetArray(strArr, 3);  testVec.Resize(20);  printIntVec(testVec); | Length = 3  Size = 20  NOT NULL  Element 0 is: Apples  Element 1 is: Oranges  Element 2 is: Pears | Y |