|  |  |
| --- | --- |
| **Ex12-03.cpp:** *Smart Pointer* | |
| **Line#** | **Code** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <iostream>  using namespace std;  #include "Student.h"  //What's wrong with this code?  int mainX() {  Student\* pAli = new Student("Ali", 21, 3.14F);  cout << "Name:" << pAli->Name << endl;  cout << "Age:" << (int)pAli->Age << endl;  cout << "CGPA:" << (\*pAli).CGPA << endl;  cout << "main() is ending..." << endl;  return 0;  } |

|  |  |
| --- | --- |
| **Ex12-03a.cpp** | |
| **Line#** | **Code** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | /\* ---- Notes ----------  -Smart pointer is a class which wraps a raw pointer, to manage the life cycle of that pointer  -It provides operator overloadings for accessing to that pointer  -The most fundamental purpose is to deal with Meory Leaking issue  -It make sure that the allocated buffer eventually will be released if no more reference to it  \*/  #include <iostream>  using namespace std;  #include "Student.h"  class MyStudent {  private:  Student\* student;  public:  explicit MyStudent(Student\* ps = nullptr) { student = ps; }  ~MyStudent() {  delete student;  }  Student& operator\*() { return \*student; }  Student\* operator->() { return student; }  };  int main() {  //Student\* pAli = new Student("Ali", 21, 3.14F);  MyStudent pAli(new Student("Ali", 21, 3.14F));  cout << "Name:" << pAli->Name << endl;  cout << "Age:" << (int)pAli->Age << endl;  cout << "CGPA:" << (\*pAli).CGPA << endl;  cout << "main() is ending..." << endl;  return 0;  } |

|  |  |
| --- | --- |
| **Ex12-03b.cpp** | |
| **Line#** | **Code** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #include <iostream>  using namespace std;  #include "Student.h"  template <class T>  class SmartPtr {  T\* ptr;  public:  explicit SmartPtr(T\* p = NULL) { ptr = p; }  ~SmartPtr() { delete (ptr); }  T& operator\*() { return \*ptr; }  T\* operator->() { return ptr; }  };  int main() {  SmartPtr<Student> pAli(new Student("Ali", 21, 3.14F));  cout << "Name:" << pAli->Name << endl;  cout << "Age:" << (int)pAli->Age << endl;  cout << "CGPA:" << (\*pAli).CGPA << endl;  cout << "main() is ending..." << endl;  return 0;  } |

|  |  |
| --- | --- |
| **Ex12-03c.cpp** | |
| **Line#** | **Code** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | /\*  unique\_ptr stores one pointer only.  We can assign a different object by removing the current object  from the pointer.  \*/  #include <iostream>  using namespace std;  #include "Student.h"  int main() {  unique\_ptr<Student> pStudent1(new Student("Ali", 21, 3.14F));  cout << \_\_LINE\_\_<<":Name:" << pStudent1->Name << endl;  pStudent1.reset(new Student("Abu", 22, 2.14F));  cout << \_\_LINE\_\_<<":Name:" << pStudent1->Name << endl;    unique\_ptr<Student> pStudent2 = move(pStudent1);  cout << \_\_LINE\_\_<<":Name:" << pStudent2->Name << endl;  unique\_ptr<Student> pStudent3(new Student("Azizi", 23, 2.14F));  pStudent2.swap(pStudent3);  cout << \_\_LINE\_\_ << ":Name:" << pStudent2->Name << endl;  cout << \_\_LINE\_\_ << ":Name:" << pStudent3->Name << endl;  cout << "main() is ending..." << endl;  return 0;  } |

|  |  |
| --- | --- |
| **Ex12-03d.cpp** | |
| **Line#** | **Code** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\*  By using shared\_ptr more than one pointers can point to this one object  at a time and it’ll maintain a Reference Counter using use\_count() method.  \*/  #include <iostream>  using namespace std;  #include "Student.h"  int main() {  shared\_ptr<Student> pStudent1(new Student("Ali", 21, 3.14F));  shared\_ptr<Student> pStudent2 = pStudent1;  cout << \_\_LINE\_\_ << ":Name:" << pStudent1->Name << endl;  cout << \_\_LINE\_\_ << ":Name:" << pStudent2->Name << endl;  cout << \_\_LINE\_\_ << ":Use Count is " << pStudent1.use\_count() << endl;  cout << "main() is ending..." << endl;  return 0;  } |