|  |
| --- |
| **Ex12-04a.cpp** |
| **Line#** | **Code** |
| 12345678910111213141516171819202122 | /\* ---- Notes -----------unique\_ptr is a class template-It is one of the smart pointer provided by C++ 11-It wrap a raw pointer in it, and de-allocate the raw pointer when unique\_ptr go out of scope.-It provides operator overloadings to simulate the raw pointer-It supports single object and list of objects\*/#include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { Student\* pAli = new Student("Ali", 21, 3.14F); //unique\_ptr<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;} |

|  |
| --- |
|  |
| **Ex12-04b.cpp** |
| **Line#** | **Code** |
| 1234567891011121314151617181920 | #include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { unique\_ptr<Student> pAli(new Student("Ali", 21, 3.14F)); unique\_ptr<Student> pWho = make\_unique<Student>(); unique\_ptr<Student> pAbu = make\_unique<Student>("Abu", 21, 3.14F); unique\_ptr<Student[]> pStudents = make\_unique<Student[]>(5); float cgpa = pStudents[2].CGPA; Student\* pAhmad = new Student("Ahmad", 22, 2.14F); unique\_ptr<Student> pAhmad1(pAhmad); //unique\_ptr<Student> pAhmad2(pAhmad); cout << "main() is ending..." << endl; return 0;} |

|  |
| --- |
|  |
| **Ex12-04c.cpp** |
| **Line#** | **Code** |
| 123456789101112131415 | #include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { unique\_ptr<Student> pAli(new Student("Ali", 21, 3.14F)); //unique\_ptr<Student> pWho = pAli; //FAIL! Cannot copy ownership unique\_ptr<Student> pWho = std::move(pAli); //Moving ownership is allowed cout << (pAli ? "pAli is NOT NULL" : "pAli is NULL") << endl; cout << "Name:" << pWho->Name << endl; cout << "main() is ending..." << endl; return 0;} |

|  |
| --- |
|  |
| **Ex12-04d.cpp** |
| **Line#** | **Code** |
| 1234567891011121314 | #include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { unique\_ptr<Student> pAli(new Student("Ali", 21, 3.14F)); Student\* pStudent = pAli.get(); cout << (pAli ? "pAli is NOT NULL" : "pAli is NULL") << endl; cout << "Name:" << pStudent->Name << endl; cout << "main() is ending..." << endl; return 0;} |

|  |
| --- |
|  |
| **Ex12-04e.cpp** |
| **Line#** | **Code** |
| 1234567891011121314 | #include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { unique\_ptr<Student> pAli(new Student("Ali", 21, 3.14F)); Student\* pStudent = pAli.release(); cout << (pAli ? "pAli is NOT NULL" : "pAli is NULL") << endl; cout << "Name:" << pStudent->Name << endl; cout << "main() is ending..." << endl; return 0;} |

|  |
| --- |
|  |
| **Ex12-04f.cpp** |
| **Line#** | **Code** |
| 123456789101112131415 | #include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { unique\_ptr<Student> pStudent(new Student("Ali", 21, 3.14F)); Student\* pAbu = new Student("Abu", 22, 2.14F); cout << "Name:" << pStudent->Name << endl; pStudent.reset(pAbu); cout << "Name:" << pStudent->Name << endl; cout << "main() is ending..." << endl; return 0;} |

|  |
| --- |
|  |
| **Ex12-04g.cpp** |
| **Line#** | **Code** |
| 1234567891011121314151617 | #include <iostream>#include <memory>using namespace std;#include "Student.h"int main() { unique\_ptr<Student> pStudent1(new Student("Ali", 21, 3.14F)); unique\_ptr<Student> pStudent2(new Student("Abu", 22, 2.14F)); cout << "Student 1:" << pStudent1->Name << '\t' << "Student 2:" << pStudent2->Name << endl; pStudent1.swap(pStudent2); cout << "Student 1:" << pStudent1->Name << '\t' << "Student 2:" << pStudent2->Name << endl; cout << "main() is ending..." << endl; return 0;} |