**DANH SÁCH LIÊN KẾT**

**Áp dụng – Nâng cao**

1. Bổ sung chương trình mẫu cho phép tính **tổng giá trị** các phần tử trên danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm Travel để viết hàm **SumList**.

|  |
| --- |
| /// File: linkedlist.cpp  ...  int linkedlist::SumList()  {  int sum = 0;  element\* p = this->head;  while (p != nullptr) {  sum += p->Getdata();  p = p->Getpointer();  }  return sum;  }  . . . |

1. Bổ sung chương trình mẫu cho phép tìm **giá trị nguyên lớn nhất** trong số các phần tử nguyên trên danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm Travel để viết hàm **MaxList**.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  int linkedlist::MaxList()  {  element\* p = this->head;  int \_max = p->Getdata();  while (p != nullptr) {  \_max = max(p->Getdata(), \_max);  p = p->Getpointer();  }  return \_max;  }  . . . |

1. Bổ sung chương trình mẫu cho phép tính **số lượng các phần tử là số nguyên tố** của danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm Travel để viết hàm **CountPrime**.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  int linkedlist::CountPrime()  {    element\* p = this->head;  int cnt = 0;  while (p != nullptr) {  int d = p->Getdata();  int check = false;  if (d < 2) check = false;  else if (d == 2) check = true;  else {  int flag = 0;  check = true;  for (int i = 2; i < sqrt(d); i++) {  if (d % i == 0) {  check = false;  break;  }  }  }  if (check) cnt++;  p = p->Getpointer();  }  return cnt;  }  . . . |

1. Bổ sung chương trình mẫu cho phép **thêm vào cuối** danh sách liên kết đơn một giá trị nguyên.

Gợi ý: tham khảo hàm InsertFirst để viết hàm Insert**Tail**.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  int linkedlist::MaxList()  {  element\* p = this->head;  int \_max = p->Getdata();  while (p != nullptr) {  \_max = max(p->Getdata(), \_max);  p = p->Getpointer();  }  return \_max;  }  . . . |

1. Bổ sung chương trình mẫu cho phép **thêm phần tử vào sau p(tham số truyền vào là 1 con trỏ)**  danh sách liên kết đơn một giá trị nguyên

|  |
| --- |
| /// File: linkedlist.cpp  . . .  void linkedlist::InsertAfterP(element\* p)  {  element\* insertE;  int temp\_data;  cout << "Nhap gia tri muon them :"; cin >> temp\_data;  insertE->Setdata(temp\_data);  if (p == this->head) {  return this->InsertFirst(insertE);  }  else if (p == this->tail) {  return this->InsertTail(insertE);  }  else {  element\* cur = this->head->Getpointer();  while (cur != nullptr || cur != p) {  cur = cur->Getpointer();  }  if (cur == nullptr) {  return;  }  insertE->Setpointer(p->Getpointer());  p->Setpointer(insertE);  return;  }  }  . . . |

1. Bổ sung chương trình mẫu cho phép **xóa phần tử đầu** danh sách liên kết đơn.
2. Bổ sung chương trình mẫu cho phép **xóa phần tử cuối** danh sách liên kết đơn.
3. Bổ sung chương trình mẫu cho phép **xóa phần tử p(tham số truyền vào là 1 con trỏ) ở vị trí bất kỳ** danh sách liên kết đơn.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  void linkedlist::DeleteP(element\* p)  {  if (p == this->head) {  this->DeleteFirst();  return;  }  else {  element\* after = this->head;    while (after->Getpointer() != p) {  after = after->Getpointer();  }  element\* cur = after->Getpointer();  after->Setpointer(cur->Getpointer());  delete cur;  return;  }  }  . . . |

1. Bổ sung chương trình mẫu cho biết **số lượng các phần tử** trên danh sách liên kết đơn có giá trị trùng với giá trị ***x*** được cho trước.

Gợi ý: tham khảo thao tác duyệt danh sách liên kết trong hàm **Travel**.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  int linkedlist::ValueDuplicate(int x)  {  int cnt = 0;  element\* p = this->head;  while (p != nullptr) {  if (p->Getdata() == x) {  cnt++;  };  p = p->Getpointer();  }  return cnt;  }  . . . |

1. Bổ sung chương trình mẫu cho phép tạo một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên trong đó không có cặp phần tử nào mang giá trị giống nhau.

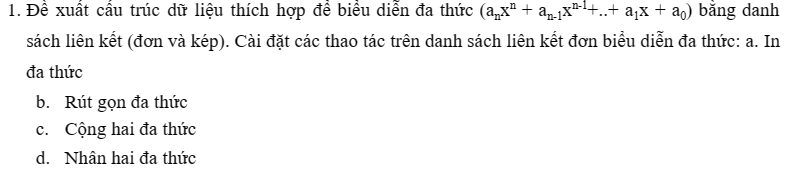
Gợi ý: sử dụng hàm InsertFirst hoặc InsertTail có bổ sung thao tác kiểm tra phần tử giống nhau.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  void linkedlist::InsertFirst(element\* e){  if (this->ValueDuplicate(e->Getdata()) > 0) return;  if(this->head==nullptr)  this->head=this->tail=e;  else{  e->Setpointer(this->head);  this->head=e;  }  this->nNum++;  }  . . .  void linkedlist::InsertTail(element\*e){  if (this->ValueDuplicate(e->Getdata()) > 0) return;  if(this->head==nullptr)  this->head=this->tail=e;  else{  this->tail->Setpointer(e);// step 1  this->tail=e; // step 2  }  this->nNum++;  }. . . |

1. Cho sẵn một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên và một giá trị nguyên ***x***. Hãy tách danh sách liên kết đã cho thành 2 danh sách liên kết: một danh sách gồm các phần tử có giá trị nhỏ hơn giá trị ***x*** và một danh sách gồm các phần tử có giá trị lớn hơn giá trị ***x***.

|  |
| --- |
| /// File: linkedlist.cpp  . . .  void linkedlist::SplitLinkedList(linkedlist\* listA, linkedlist\* listB, int x)  {  element\* p = this->head;  while (p != nullptr) {  if (p->Getdata() > x) {  listA->InsertTail(p);  }  else if (p->Getdata() < x) {  listB->InsertTail(p);  }  p = p->Getpointer();  }  return;  }. . |

**Bài tập Ứng dụng**



|  |
| --- |
| //DaThuc.h  #include "PhanTu.h"  #pragma once  class DaThuc  {  private:  PhanTu\* head;  PhanTu\* tail;  int nNum;  public:  DaThuc();  PhanTu\* Gethead() { return head; }  PhanTu\* Gettail() { return tail; }  void Sethead(PhanTu\*);  void Settail(PhanTu\*);  void Insert(PhanTu\*);  void Delete(PhanTu\*);  void In();  void Rutgon();  DaThuc\* Cong(DaThuc\* dt2);  DaThuc\* Nhan(DaThuc\* dt2);  };  //DaThuc.cpp  #include "DaThuc.h"  #include <iostream>  using namespace std;  DaThuc::DaThuc()  {  this->head = nullptr;  this->tail = nullptr;  this->nNum = 0;  }  void DaThuc::Sethead(PhanTu\* \_head)  {  this->head = \_head;  return;  }  void DaThuc::Settail(PhanTu\* \_tail)  {  this->tail = \_tail;  return;  }  void DaThuc::Insert(PhanTu\* \_next)  {  if (this->head == nullptr) {  this->head = this->tail = \_next;  return;  }  this->tail->GanNext(\_next);  this->tail = \_next;  this->tail->GanNext(nullptr);  }  void DaThuc::Delete(PhanTu\* target)  {  if (this->head == target) {  this->head = this->head->LayNext();  return;  }  PhanTu\* p = this->head;  while (p != nullptr)  {  if (p->LayNext() == target) {  p->GanNext(target->LayNext());  return;  }  p = p->LayNext();  }  return;  }  void DaThuc::In()  {  PhanTu\* p = this->head;  while (p != nullptr)  {  cout << p->LayCoso() << "\*X^" << p->LaySomu();  p = p->LayNext();  if (p != nullptr) cout << " + ";  }  return;  }  void DaThuc::Rutgon()  {  PhanTu\* p = this->head;  while (p != nullptr)  {  PhanTu\* p2 = p->LayNext();  while (p2 != nullptr)  {  PhanTu\* tmp = p2->LayNext();  if (p2->LaySomu() == p->LaySomu()) {  p->GanCoso(p->LayCoso() + p2->LayCoso());  this->Delete(p2);  }  p2 = tmp;  }  p = p->LayNext();  }  return;  }  DaThuc\* DaThuc::Cong(DaThuc\* dt2)  {  DaThuc\* tong = new DaThuc();  PhanTu\* p1 = this->head;  while (p1 != nullptr)  {  tong->Insert(p1);  p1 = p1->LayNext();  }  PhanTu\* p2 = dt2->head;  while (p2 != nullptr)  {  tong->Insert(p2);  p2 = p2->LayNext();  }  tong->Rutgon();  return tong;  }  DaThuc\* DaThuc::Nhan(DaThuc\* dt2)  {  DaThuc\* tich = new DaThuc();  PhanTu\* p1 = this->head;  while (p1 != nullptr)  {  PhanTu\* p2 = this->head;  while (p2 != nullptr) {  if (p2->LaySomu() == p1->LaySomu()) {  PhanTu\* p3 = new PhanTu();  p3->GanCoso(p1->LayCoso() \* p2->LayCoso());  p3->GanSomu(p1->LaySomu() \* p2->LaySomu());  tich->Insert(p3);  }  p2 = p2->LayNext();  }  }  tich->Rutgon();  return tich;  }  //PhanTu.h  #pragma once  class PhanTu  {  private:  int coso;  int somu;  PhanTu\* next;  public:  PhanTu();  void GanCoso(int);  void GanSomu(int);  void GanNext(PhanTu\*);  int LayCoso();  int LaySomu();  PhanTu\* LayNext();  };  //PhanTu.cpp  #include "PhanTu.h"  #include <iostream>  using namespace std;  PhanTu::PhanTu()  {  this->coso = NULL;  this->somu = NULL;  this->next = nullptr;  }  void PhanTu::GanCoso(int \_coso)  {  this->coso = \_coso;  return;  }  void PhanTu::GanSomu(int \_somu)  {  this->somu = \_somu;  }  void PhanTu::GanNext(PhanTu\* \_next)  {  this->next = \_next;  return;  }  int PhanTu::LayCoso()  {  return this->coso;  }  int PhanTu::LaySomu()  {  return this->somu;  }  PhanTu\* PhanTu::LayNext()  {  return this->next;  } |