PHẦN 1: DANH SÁCH LIÊN KẾT

MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

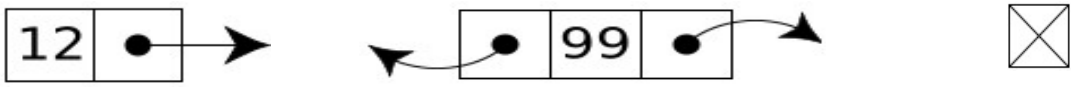
- Hiểu được các thành phần của danh sách liên kết.
- Thành thạo các thao tác trên danh sách liên kết: thêm phần tử, xóa phần tử, duyệt danh sách liên kết.
- Áp dụng cấu trúc dữ liệu danh sách liên kết vào việc giải quyết một số bài toán đơn giản. Thời gian thực hành: **từ 120 phút đến 400 phút**

TÓM TẮT

Danh sách liên kết là cấu trúc dữ liệu dùng để lưu trữ một danh sách (tập hợp hữu hạn) dữ liệu. Điểm đặc biệt của cấu trúc này là khả năng chứa của nó **động** (có thể mở rộng và thu hẹp dễ dàng). Có các loại danh sách liên kết:

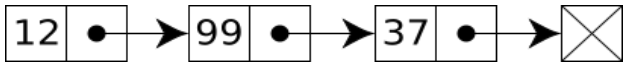
- Danh sách liên kết đơn
- Danh sách liên kết kép
- Danh sách liên kết vòng

Mỗi danh sách liên kết là tập hợp các phần tử (element) chứa thông tin lưu trữ của dữ liệu. Giữa các phần tử có một hoặc nhiều liên kết để đảm bảo danh sách liên kết có thể giữ các phần tử này một cách chặt chẽ. *Ví dụ 1:*

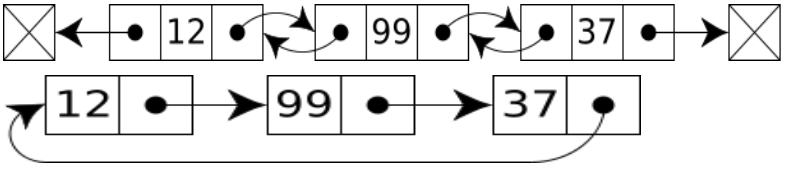


Phần tử có một liên kết | Phần tử có hai liên kết | Phần tử rỗng

Ví dụ 2:



Danh sách liên kết đơn



Danh sách liên kết kép

Danh sách liên kết vòng

Trong mỗi phần tử của danh sách liên kết, thông tin liên kết là vô cùng quan trọng. Chỉ cần một xử lý không cẩn thận có thể làm mất phần liên kết này thì danh sách liên kết sẽ bị '**gãy**' từ phần tử đó (không thể truy xuất tiếp các phần tử từ phần tử đó trở về trước hoặc trở về sau).

Các thao tác cơ bản trên danh sách liên kết:

- Thêm phần tử: vào đầu danh sách liên kết, vào cuối danh sách liên kết, vào trước/sau một phần tử trên danh sách liên kết.
- Xóa phần tử: ở đầu danh sách liên kết, ở cuối danh sách liên kết, một phần tử trên danh sách liên kết.
- Duyệt danh sách liên kết: để có thể đi được hết các phần tử trên danh sách liên kết.

NỘI DUNG THỰC HÀNH

Cơ bản

Sinh viên đọc kỹ phát biểu bài tập và thực hiện theo hướng dẫn:

Tổ chức một danh sách liên kết đơn trong đó mỗi phần tử chứa thông tin dữ liệu nguyên.

Phân tích

- Danh sách liên kết đơn gồm mỗi phần tử chứa dữ liệu nguyên. Thông tin của mỗi phần tử được khai báo theo ngôn ngữ C/C++ như sau:
- Thao tác cần thực hiện: **thêm** phần tử nguyên **vào đầu** danh sách liên kết (**First**), **in** các phần tử của danh sách liên kết (**Travel**), **loại bỏ tất cả** các phần tử trên danh sách liên kết (**RemoveAll**).

Chương trình mẫu

```
/// File: element.h
#ifndef ELEMENT_H
#define ELEMENT_H


class element
{
   private:
      int data;
      element *pointer;

   public:
      element();
      element(int);
      virtual ~element();

      int Getdata() { return data; }
      void Setdata(int val) { data = val; }
      element * Getpointer() { return pointer; }
      void Setpointer(element* val) { pointer = val; }

   protected:

};

#endif // ELEMENT_H
```

```
/// File: element.cpp
#include "element.h"

element::element()
{
    //ctor
    this->data=0;
    this->pointer=nullptr;
}
element::element(int data)
{
    //ctor
    this->data=data;
    this->pointer=nullptr;
}

element::~element()
{
    //dtor
}


/// File: linkedlist.h
#ifndef LINKEDLIST_H
#define LINKEDLIST_H
#include "element.h"

class linkedlist
{
     private:
        element* head;
        element* tail;
        int nNum;
    public:
        linkedlist();
        virtual ~linkedlist();
        element* Gethead() { return head; }
        void Sethead(element* val) { head = val; }
        element* Gettail() { return tail; }
        void Settail(element* val) { tail = val; }
        void InsertFirst(element*);
        void InsertTail(element*);
        bool DeleteFirst();
        void Travel();

    protected:


};

#endif // LINKEDLIST_H


/// File: linkedlist.cpp

#include "linkedlist.h"
#include <iostream>
using namespace std;
```

```
linkedlist::linkedlist()
{
    //ctor
    this->head=nullptr;
    this->tail=nullptr;
    this->nNum=0;
}

linkedlist::~linkedlist()
{
    //dtor
}
void linkedlist::InsertFirst(element* e){
      if(this->head==nullptr)
        this->head=this->tail=e;
        else{
            e->Setpointer(this->head); //step 1
            this->head=e;  // step 2
        }
       this->nNum++;

}
 void linkedlist::InsertTail(element*e){
     if(this->head==nullptr)
        this->head=this->tail=e;
        else{
            this->tail->Setpointer(e);// step 1
            this->tail=e;    // step 2
        }
     this->nNum++;
 }
void linkedlist::Travel(){
     element* p=this->head;
     while(p!=nullptr){
        cout<<p->Getdata()<<"\t";
        p=p->Getpointer();
     }
}
  bool linkedlist::DeleteFirst(){
      if(this->head==nullptr) return false;
      else{
      element*p=this->head;
      this->head=this->head->Getpointer();
      delete p;
      return true;
      }
  }
```

///File: main.cpp

```
#include <iostream>
#include "linkedlist.h"
using namespace std;

int main()
{
    linkedlist *list_=new linkedlist();
```

```
    element *e;
    e=new element(9);
    list_->InsertTail(e);
    e=new element(10);
     list_->InsertTail(e);
    e=new element(8);
     list_->InsertTail(e);
    list_->Travel();
    list_->DeleteFirst();
    cout<<"\n";
     list_->Travel();
    return 0;
}
```

Yêu cầu

1. Biên dịch đoạn chương trình nêu trên.
2. Cho biết kết quả in ra màn hình:
3. Nêu nhận xét ngắn gọn mối liên hệ giữa thứ tự nhập dữ liệu vào với thứ tự in dữ liệu ra màn hình.
4. Vẽ hình danh sách liên kết theo dữ liệu được nhập ở câu 2.
5. Viết hàm RemoveAll xóa tất cả các phần tử trong DSLK

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**.

2. 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**.

3. 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**.

4. 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**.

5. 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

6. Bổ sung chương trình mẫu cho phép **xóa phần tử đầu** danh sách liên kết đơn.

7. Bổ sung chương trình mẫu cho phép **xóa phần tử cuối** danh sách liên kết đơn.

8. 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.

9. 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**.

10. 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.

11. 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***.

BÀI TẬP ỨNG DỤNG

1. Đề xuất cấu trúc dữ liệu thích hợp để biểu diễn đa thức ($a_nx^n + a_{n-1}x^{n-1}+..+ a_1x + a_0$) bằng danh sách liên kết (đơn và kép). Cài đặt các thao tác trên danh sách liên kết đơn biểu diễn đa thức: a. In đa thức
   b. Tính giá trị đa thức (với giá trị x nhập vào)
   c. Cộng hai đa thức
   d. Nhân hai đa thức

2. Thông tin của một quyển sách trong thư viện gồm các thông tin:
   - Tên sách (chuỗi)
   - Tác giả (chuỗi, tối đa 5 tác giả)
   - Nhà xuất bản (chuỗi)
   - Năm xuất bản (số nguyên)

a. Hãy tạo danh sách liên kết (đơn hoặc kép) chứa thông tin các quyển sách có trong thư viện (được nhập từ bàn phím).
b. Cho biết số lượng các quyển sách của một tác giả bất kỳ (nhập từ bàn phím).
c. Trong năm YYYY (nhập từ bàn phím), nhà xuất bản ABC (nhập từ bàn phím) đã phát hành những quyển sách nào.

PHẦN 2: STACK - QUEUE

MỤC TIÊU

Hoàn tất phần thực hành này, sinh viên có thể:

- Hiểu được cách thức sử dụng stack và queue trên cơ sở sử dụng danh sách liên kết để cài đặt.
- Hiểu và vận dụng các cấu trúc stack và queue trong những bài toán đơn giản.

Thời gian thực hành: 120 phút đến 360 phút.

Lưu ý: yêu cầu vận dụng thành thạo danh sách liên kết ở Lab01.

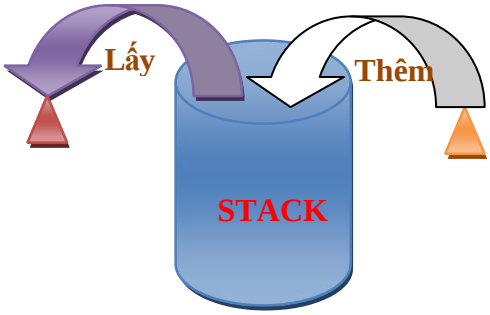
TÓM TẮT

- Stack (ngăn xếp) và queue (hàng đợi) là những cấu trúc dữ liệu dùng để lưu trữ các phần tử của tập hợp theo những nguyên tắc đặc biệt khi thêm phần tử cũng như lấy phần tử ra khỏi cấu trúc.
- Stack (last in, first out – LIFO): phần tử vào stack sau cùng, là phần tử được lấy ra khỏi stack trước nhất.
- Queue (first in, first out – FIFO): phần tử vào queue trước nhất, là phần tử được lấy ra khỏi queue trước nhất.

☺ Lab03 là phần vận dụng danh sách liên kết đã thực hành ở Lab02 để cài đặt Stack và Queue.

(Lưu ý: chúng ta cũng có thể dùng mảng để cài đặt stack và queue, những mảng đặc trưng cho cơ chế tĩnh, do vậy danh sách liên kết – cơ chế động - là cấu trúc tốt hơn mảng khi hiện thực Stack và Queue).

Ví dụ: minh họa Stack



+ Phần tử mới được thêm vào đỉnh của ngăn xếp.

+ Thao tác lấy phần tử ra khỏi ngăn xếp, nếu ngăn xếp khác rỗng thì phần tử ở đầu ngăn xếp được lấy ra, ngược lại, ngăn xếp rỗng thì thao tác lấy phần tử thất bại.

Ví dụ: minh họa Queue

+ Phần tử được thêm vào ở đầu queue. Do vậy, phần tử vào đầu tiên sẽ ở đáy của queue. Do vậy, khi lấy phần tử ra, nếu queue khác rỗng thì phần tử ở đáy queue được lấy ra, ngược lại, queue bị rỗng thì thao tác lấy phần tử ra khỏi queue thất bại.

NỘI DUNG THỰC HÀNH

Cơ bản

Yêu cầu: cài đặt stack và queue bằng danh sách liên kết.

Do đặc trưng của stack và queue, chúng ta cần xây dựng 2 thao tác chính là thêm 1 phần tử vào stack hoặc queue, và lấy 1 phần tử ra khỏi stack hoặc queue.

Dựa vào nguyên tắc thêm và lấy phần tử ra khỏi stack/queue, ta cần xây dựng các hàm sau:

- Đối với Stack o Thêm phần tử: thêm phần tử vào đầu danh sách liên kết.
    - Lấy phần tử: lấy phần tử ở đầu danh sách ra khỏi danh sách liên kết.

(Lưu ý: ta cũng có thể thêm phần tử vào cuối danh sách liên kết, do vậy thao tác lấy phần tử, ta thực hiện lấy phần tử ở cuối danh sách liên kết).

- Đối với Queue o Thêm phần tử: thêm vào đầu danh sách liên kết.
    - Lấy phần tử: lấy phần tử ở cuối danh sách liên kết.

(Lưu ý: ta cũng có thể thực hiện việc thêm phần tử vào cuối danh sách liên kết và lấy ra ở đầu danh sách liên kết).

```
#ifndef STACK_H
#define STACK_H
#include "LinkedList.h"

class Stack
{
   private: int nNum;
            LinkedList *linkedlist;
   public:
      Stack();
      virtual ~Stack();
      void push(int);
      int pop();
};

#endif // STACK_H

#include "Stack.h"

 void Stack::push(int x){
    Element *p=new Element(x);
```

```
    this->linkedlist->insertFirst(p);
 }
 int Stack::pop(){
   int p=this->linkedlist->getHead()->getdata();
   this->linkedlist->deletefirst();
   return p;

 }
void Stack::printStack(){
    // Viết tiếp code
  }
Stack::Stack()
{
   //ctor
   this->nNum=0;
   this->linkedlist=new LinkedList();
}
Stack::~Stack()
{
   //dtor
}

int main()
{

   Stack *s=new Stack();
   s->push(7);
   s->push(10);
   s->push(18);
   s->push(20);
   s->prinStack();
   s->pop();
   cout<<"\n";
   s->printStack();

   return 0;
}
```

1. Sử dụng danh sách liên kết để cài đặt cấu trúc Stack, Queue.
2. Sử dụng các hàm PushStack, PopStack, EnQueue, DeQueue để cài đặt.
    a. Về Stack: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lệnh thêm phần tử vào stack), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi stack, nếu có, in giá trị phần tử ra màn hình, nếu không có (stack rỗng), in ra màn hình "STACK RONG, KHONG LAY DUOC PHAN TU".
    b. Về Queue: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lần lệnh thêm phần tử vào queue), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi queue, nếu có, in giá trị phần tử ra màn hình, nếu không có (queue rỗng), in ra màn hình "QUEUE RONG, KHONG LAY DUOC PHAN TU".

Áp dụng – Nâng cao

1. Sử dụng hàm InsertTail và DeleteTail trong LinkedList để có phiên bản cài đặt Stack (thêm phần tử vào cuối danh sách và lấy phần tử ở cuối danh sách liên kết) cũng như áp dụng 1 phiên bản khác khi cài đặt Queue (thêm phần tử vào cuối danh sách liên kết và lấy phần tử ở đầu danh sách liên kết).
2. Nhận xét cách cài đặt trên ở phần 1 (áp dụng – nâng cao) so với chương trình mẫu đối với trường hợp stack cũng như queue.
3. Sử dụng cấu trúc Stack để chuyển giá trị từ cơ số 10 sang cơ số 2.
   Gợi ý : thực hiện việc chia liên tiếp giá trị trong cơ số 10 cho 2, lấy phần dư đưa vào stack, cho đến khi giá trị đem đi chia là 0. In giá trị trong stack ra (đó chính là kết quả khi chuyển số từ hệ cơ số 10 sang hệ cơ số 2).

BÀI TẬP ỨNG DỤNG

Bài 1: Tìm đường trong mê cung (thực hiện loang theo chiều rộng <sử dụng queue> hoặc loang theo chiều sâu <sử dụng stack>).

Bài toán: cho ma trận mxn, mỗi phần tử là số 0 hoặc 1.

Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được.

Câu hỏi:

Từ ô ban đầu có tọa độ (x1, y1) có thể đi tới ô (x2, y2) không?

Biết rằng từ 1 ô (x,y) chỉ có thể đi qua ô có chung cạnh với ô đang đứng và mang giá trị là 1, ngược lại không có đường đi.

Bài 2: Bài toán di chuyển toa tàu (hình dưới): Các toa được đánh số từ 1 đến n, đường di chuyển có thể là các vạch đỏ. Ta cần di chuyển các toa từ A -> C sao cho tại C các toa tàu được sắp xếp các thứ tự mới nào đó. Hãy nhập vào thứ tự tại C cần có, cho biết có cách chuyển không ? Nếu có, hãy trình bày cách chuyển.

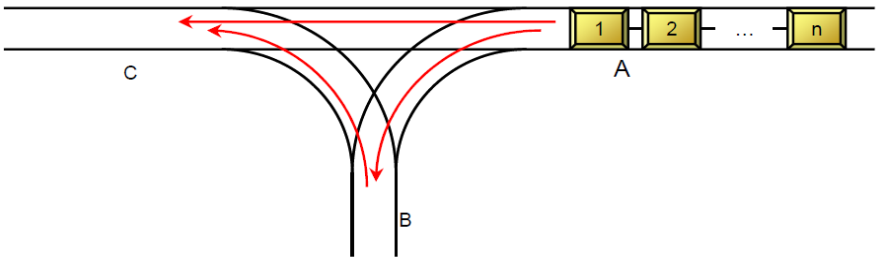
Ví dụ: n = 4 và thứ tự cần có (1, 4, 3, 2)

1) A->C
2) A->B
3) A->B

4) A->C

5) B->C

6) B->C



Bài 3: Tương tự yêu cầu bài 3 nhưng với hình bên dưới: