REPORT_24120184_WEEK4

Linkedlist:

1. NODE* createNode(int data):

-Mục đích hàm:

- Tạo 1 một node mới có giá trị là data trong danh sách liên kết đơn(với data là giá trị được truyền vào hàm).

-Ý tưởng viết hàm:

- Khai báo 1 con trỏ có kiểu dữ liệu là NODE
- Cho p_next của node mới vừa tạo trỏ đến nullptr.(là node tiếp theo của node mới).
- Gán key của node mới bằng data.
- Trả về con trỏ kiểu NODE.

2. List* createList(NODE* p_node):

-Mục đích của hàm:

- Tạo 1 danh sách liên kết có node đầu tiên là p_node (con trỏ p_node có kiểu dữ liệu NODE được truyền vào hàm).

-Ý tưởng viết hàm:

- Kiểm tra p_node truyền vào , nếu trỏ nullptr thì trả về con trỏ trỏ đến nullptr.(Node rỗng không tạo được danh sách mới).
- Khai báo con trỏ mới có kiểu dữ liệu là List.
- Cho con trỏ p_head = p_node , p_tail = p_node (vì là phần tử đầu tiên của danh sách).
- Cho con trỏ p_next của p_head trỏ đến nullptr (đề phòng p_next của p_node không trỏ đến nullptr).
- Trả về con trỏ trỏ đến danh sách vừa mới khởi tạo.

3. bool addHead(List* &L, int data):

-Mục đích của hàm:

- Thêm vào đầu danh sách liên kết 1 node có giá trị là data (truyền vào hàm con trỏ của danh sách liên kết, giá trị của data).

-Ý tưởng viết hàm:

- Khai báo 1 con trỏ mới newnode bằng hàm createNode có giá trị là data.
- Nếu danh sách trống hay p_head bằng nullptr , cho p_head của danh sách liên kết bằng newnode và p_tail của danh sách liên kết bằng newnode, trả về true.
- Nếu danh sách không trống thì cho newnode trỏ đến p_head (Chèn newnode là node đầu tiên của danh sách) và cho p_head trỏ đến newnode để cập nhật vị trí p_head, trả về true.

4. bool addTail(List* &L, int data):

-Mục đích của hàm:

- Thêm vào cuối danh sách liên kết 1 node có giá trị là data (truyền vào hàm con trỏ của danh sách liên kết, giá trị của data).

-Ý tưởng của viết hàm:

- Khai báo 1 con trỏ mới newnode bằng hàm createNode có giá trị là data.
- Nếu danh sách trống hay p_head bằng nullptr , cho p_head của danh sách liên kết bằng newnode và p_tail của danh sách liên kết bằng newnode, trả về true.
- Nếu danh sách không trống thì cho newnode trỏ đến p_tail và cho p_tail trỏ đến newnode để cập nhật vị trí p_head, trả về true.

5. bool removeHead(List* &L):

-Mục đích của hàm:

- Xóa node đầu của danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Nếu p_head của danh sách bằng nullptr(danh sách liên kết trống) thì trả về false.
- Nếu danh sách liên kết có 1 phần tử thì kiểm tra nếu con trỏ p_head của danh sách liên kết trỏ đến nullptr thì cho cho p_tail của danh sách liên kết bằng nullptr.
- Khai báo 1 con trỏ temp có kiểu dữ liệu NODE bằng p_head , cho p_head trỏ đến node tiếp theo , xóa node temp vừa khai báo.
- Trả về true.

6. void removeTail(List *&L)

-Mục đích của hàm:

- Xóa node cuối của danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Nếu p_head của danh sách bằng nullptr(danh sách liên kết trống) thì trả về false.
- Nếu danh sách liên kết có 1 phần tử thì kiểm tra nếu con trỏ p_head của danh sách liên kết trỏ đến nullptr thì cho khai báo 1 con trỏ temp có kiểu dữ liệu NODE bằng p_head, cho p_head và p_tail bằng nullptr, xóa temp thoát khỏi hàm.
- Trường hợp có 2 node trở lên:
  - Khai báo 1 con trỏ temp có kiểu dữ liệu NODE bằng p_head , kiểm tra vòng lặp cho con trỏ temp chạy đến node đứng trước p_tail.
  - Cho p_tail bằng temp , temp bằng con trỏ trỏ đến node kế của temp , cho p_tail trỏ đến nullptr , xóa con trỏ temp.

7. void removeAll(List* &L)

-Mục đích của hàm:

- Xóa tất cả node có trong danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Nếu p_head bằng nullptr(danh sách liên kết trống) thoát khỏi hàm.
- Khai báo 1 con trỏ cur có kiểu dữ liệu NODE bằng p_head.
- Duyệt qua từng node(xóa từng node):
  - Khai báo 1 con trỏ temp có kiểu dữ liệu NODE bằng cur.
  - Cho cur bằng cur->p_next (Di chuyển đến Node tiếp theo và xóa).
  - Xóa temp
- Cho p_head và p_tail bằng nullptr.

8. removeBefore(List *&L, int val)

-Mục đích của hàm:

- Xóa 1 node nằm phía trước 1 node có giá trị là val (truyền vào hàm con trỏ của danh sách liên kết, giá trị của val).

-Ý tưởng viết hàm:

- Nếu p_head bằng nullptr (danh sách rỗng) và giá trị của node đầu bằng val thì thoát khỏi hàm (do node đầu không có node phía trước).
- Nếu giá trị của node sau p_Head bằng val thì xóa node đầu qua hàm removeHead.
- Khai báo 1 con trỏ cur có kiểu dữ liệu NODE bằng p_head để duyệt danh sách.
- Duyệt cho đến khi còn ít nhất 3 node (vì cần xét node đứng trước node có giá trị val, tức là cần nhìn 2 node phía trước node val):
  - Nếu node sau node sau cur (là node thứ 3 kể từ cur) có key bằng val, thì node giữa (cur->p_next) chính là node cần xóa.
  - Khai báo con trỏ temp bằng node cần xóa.
  - Nối cur với node sau node bị xóa (bỏ qua node temp).
  - Xóa node temp và thoát khỏi hàm.

9. void romveAfter(List* &L, int val)

-Mục đích của hàm:

- Xóa 1 node nằm phía sau 1 node có giá trị là val (truyền vào hàm con trỏ của danh sách liên kết, giá trị của val).

-Ý tưởng viết hàm:

- Nếu p_head bằng nullptr (danh sách rỗng) thì thoát khỏi hàm.
- Khai báo 1 con trỏ cur có kiểu dữ liệu NODE bằng p_head để duyệt danh sách.
- Duyệt cho đến node cuối tức là cur bằng nullptr:
  - Nếu giá trị của node cur bằng val thì:

+Nếu cur đang là node cuối (sau node cuối không còn node nào để xóa) thì thoát khỏi hàm.

+Khai báo con trỏ temp bằng node kế tiếp của cur (node cần xóa).

+Cho node cur trỏ đến node thứ 3 kể từ cur (p_next của cur bằng cur->p_next->p_next).

+Nếu node cần xóa là p_tail thì cập nhật vị trí p_tail là cur.

+Xóa node temp và thoát khỏi hàm.

10. bool addPos(List* &L, int data, int pos)

-Mục đích của hàm:

- Thêm node có giá trị là data vào vị trí nhất định của danh sách liên kết(truyền vào con trỏ của danh sách liên kết, giá trị của data, vị trí cần chèn pos).

-Ý tưởng của hàm:

- Trường hợp vị trí nhỏ hơn 0 trả về false không thể chèn do vị trí không phù hợp.
- Trường hợp vị trí bằng 0 chạy hàm addHead và trả về giá trị true.
- Các trường hợp còn lại:
  - Khai báo con trỏ newnode có giá trị data bằng hàm createNode.
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Cho i bằng 0 chạy vòng lặp đến i bằng pos – 2, trong mỗi lần lặp nếu cur bằng nullptr thì xóa newnode và trả về false do vị trí chèn vượt quá số phần tử trong danh sách
  - Sau vòng lặp:

+Nếu cur bằng nullptr thì xóa newnode trả về false.(Khi cập nhật vị trí cur kiểm tra lại).

+Cho newnode trỏ đến node tiếp theo của cur (tức node tiếp theo cur là vị trí cần chèn)

+Cho cur trỏ đến newnode.

+Nếu newnode là phần tử cuối hay newnode trỏ đến nullptr thì cập nhật vị trí p_tail.

+Trả về true.

11. removePos(List *&L, int data, int pos)

-Mục đích của hàm:

- Xóa bỏ phần tử có vị trí pos của danh sách liên kết tính từ vị trí 0 (truyền vào con trỏ của danh sách liên kết, giá trị của data, vị trí cần chèn pos).

-Ý tưởng của hàm:

- Trường hợp hàm là danh sách rỗng hoặc vị trí chèn nhỏ hơn 0 thì thoát khỏi hàm.
- Trường hợp vị trí chèn là 0 thì chạy hàm removeHead.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Cho i bằng 0 chạy vòng lặp đến i bằng pos – 2, trong mỗi lần lặp nếu cur bằng nullptr thì thoát khỏi hàm do vị trí chèn vượt quá số phần tử trong danh sách
  - Sau vòng lặp:

+Nếu cur bằng nullptr thì xóa newnode thoát khỏi hàm.(Khi cập nhật vị trí cur kiểm tra lại).

+Cho con trỏ temp bằng node ở sau cur hay temp bằng pNext của cur (node temp là node cần xóa).

+Nếu temp là node cuối thì cập nhật p_tail bằng cur.

+Cho cur trỏ đến node sau node cần xóa (cập nhật vị trí của pNext của cur).

+Xóa node temp và thoát khỏi hàm.

12. bool addBefore(List *&L, int data, int val)

-Mục đích của hàm:

- Chèn Node có giá trị là data vào trước node có giá trị là val (nếu có) (truyền vào con trỏ của danh sách liên kết, giá trị của data, giá trị của val).

-Ý tưởng viết hàm:

- Trường hợp hàm là danh sách rỗng thì thoát trả về giá trị false(không chèn được).
- Trường hợp node đầu của danh sách liên kết có giá trị bằng val thì chạy hàm addHead với node cần thêm có giá trị là data, trả về true.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Chạy vòng lặp để xác định Node đứng trước Node có giá trị bằng val, trong quá trình chạy nếu tìm được Node thỏa thì thoát khỏi vòng lặp.
  - Sau vòng lặp:

+Kiểm tra xem đã chạy đến phần tử cuối chưa nếu đã chạy đến phần tử cuối tức là không có node nào có giá trị bằng val nên trả về false.

+Khai báo con trỏ của node mới newnode có giá trị là data.

+Cho newnode trỏ đến node có giá trị là val.

+Cho node cur (node đứng trước node có giá trị là val) trỏ đến newnode, trả về true.

13. bool addAfter(List* &L, int data, int val)

-Mục đích của hàm:

- Chèn Node có giá trị là data vào sau node có giá trị là val (nếu có) (truyền vào con trỏ của danh sách liên kết, giá trị của data, giá trị của val).

-Ý tưởng viết hàm:

- Trường hợp hàm là danh sách rỗng thì thoát trả về giá trị false(không chèn được).
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Chạy vòng lặp xác định Node có giá trị bằng val, trong quá trình chạy nếu tìm được Node thỏa mãn thì thoát ra khỏi vòng lặp.
  - Sau vòng lặp:

+Kiểm tra cur có bằng nullptr hay khồn nếu có tức là không có node nào có giá trị bằng val nên trả về false.

+Khai báo con trỏ của node mới newnode có giá trị là data.

+Cho newnode trỏ đến con Node nằm sau Node có giá trị là val (Node nằm sau cur).

+Cho cur (node có giá trị bằng val) trỏ đến newnode.

+Nếu newnode trỏ đến nullptr tức newnode là phần tử cuối, cập nhật p_tail bằng newnode, trả về true.

14.void printList(List *L)

-Mục đích của hàm:

- In ra giá trị của từng Node có trong danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết không gồm Node nào in ra NONE.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_head.
  - Duyệt qua lần lượt các Node có trong danh sách liên kết và in giá trị.

15. int countElements(List* L)

-Mục đích của hàm:

- Đếm số lượng Node , phần tử có trong danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết không gồm Node nào trả về 0.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_head.
  - Khai báo biến đếm bằng 0.
  - Duyệt qua lần lượt các Node có trong danh sách liên kết và tăng giá trị biến đếm lên 1.
  - Trả về giá trị đếm sau khi hết vòng lặp.

16. List* reverseList(List* L)

-Mục đích của hàm:

- Tạo ra một danh sách liên kết mới có dữ liệu ngược với danh sách ban đầu(truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách rỗng thì ta trả về 1 danh sách rỗng.
- Các trường hợp còn lại:
  - Khai báo 1 danh sách liên kết mới (Là danh sách liên kết đảo của danh sách ban đầu).

- Khai báo con trỏ cur bằng p_head của danh sách liên kết ban đầu (dùng để duyệt qua các node)
- Chạy vòng lặp để duyệt từng node:

+Với mỗi vòng lặp ta khai báo 1 node mới newnode có giá trị là data của node đang xét.

+Trường hợp danh sách liên kết đảo trống , ta gán p_head và p_tail của danh sách liên kết đảo bằng node mới.

+Ngược lại nếu không trống ta thêm node mới vào đầu danh sách liên kết.

- Sau vòng lặp duyệt qua các phần tử thì trả về danh sách liên kết đảo.

17. void removeDuplicate(List *&L)

-Mục đích của hàm:

- Loại bỏ các Node có giá trị trùng trong danh sách liên kết(truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết trống hoặc danh sách liên kết có 1 phần tử thì thoát ra khỏi hàm.
- Khai báo con trỏ cur bằng p_head để duyệt qua các phần tử.
- Chạy vòng lặp cho cur đi qua từng node cho đến khi cur là phần tử cuối cùng:
  - Khai báo biến check kiểu bool bằng true (đảm bảo tính đúng đắn trong lúc duyệt).
  - Tiếp tục chạy vòng lặp khai báo con trỏ cur1 bằng p_head chạy đến Node đứng sau cur:

+Trong lúc chạy vòng lặp nếu giá trị của cur1 bằng giá trị của cur thì tức là Node đang xét (là Node đứng sau cur) đã bị trùng giá trị với 1 trong những Node đứng trước nó

+Khi bị trùng ta cho check bằng false, khai báo 1 con trỏ t bằng Node ta đang xét (Node đứng sau cur) và xóa nó (nếu là Node cuối cập nhật lại p_tail).

- Sau vòng lặp nếu check bằng true thì ta xét đến node tiếp theo.

18. bool removeElement(List* &L, int key)

-Mục đích của hàm:

- Loai bỏ tất cả Node có giá trị là key trong danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách rỗng (p_head = nullptr) thì không có gì để xóa trả về false;
- Khai báo biến remove kiểu bool bằng false, sẽ kiểm tra có xóa phần tử nào không.
- Đầu tiên ta lặp liên tục để xóa khi phần tử đầu tiên có giá trị là key: (với điều kiện giá trị của p_head bằng key và p_head khác nullptr).
  - Khai báo con trỏ t bằng p_head, cập nhật vị trí p_head bằng node tiếp theo, cho remove bằng true khi xóa được ít nhất 1 phần từ , xóa con trỏ t.

- Kiểm tra khi xóa các phần tử đầu nếu danh sach rỗng cập nhật p_tail bằng nullptr rồi trả về kết quả remove.
- Khai báo con trỏ cur bằng p_head để duyệt danh sách, bắt đầu duyệt từ phần tử thứ hai trở đi (cur->p_next) để kiểm tra xem có node nào cần xóa nữa không:
  - Nếu node tiếp của cur có giá trị bằng key thì ta cần xóa node đó, cập nhật cur->p_next để bỏ qua node cần xóa, nếu node cần xóa là node cuối thì ta cập nhật vị trí p_tail bằng cur , sau đó ta xóa node và cho remove bằng true.
  - Ngược lại nếu node tiếp không bị xóa thì ta cập nhật cur để duyệt (cur = cur->p_next) do nếu bị xóa ta vẫn cập nhật thì sẽ bỏ qua 1 Node vì cur->p_next đã thay đổi và ta cần kiểm tra lại.
- Cuối cùng sau vòng lặp ta trả về remove.

DoublyLinkedlist:

1. d_NODE* createNode(int data):

-Mục đích hàm:

- Tạo 1 một node mới có giá trị là data trong danh sách liên kết đôi(với data là giá trị được truyền vào hàm).

-Ý tưởng viết hàm:

- Khai báo 1 con trỏ có kiểu dữ liệu là d_NODE
- Cho pNext của node mới vừa tạo trỏ đến nullptr.(là node tiếp theo của node mới).
- Cho pPrev của node mới vừa tạo trỏ đến nullptr.(là node phía sau của node mới).
- Gán key của node mới bằng data.
- Trả về con trỏ d_NODE.

2.d_ List* createList(d_NODE *p_node):

-Mục đích của hàm:

- Tạo 1 danh sách liên kết đôi có node đầu tiên là p_node (con trỏ p_node có kiểu dữ liệu d_NODE được truyền vào hàm).

-Ý tưởng viết hàm:

- Kiểm tra p_node truyền vào , nếu trỏ nullptr thì trả về con trỏ trỏ đến nullptr.(Node rỗng không tạo được danh sách mới).
- Khai báo con trỏ mới có kiểu dữ liệu là d_List.
- Cho con trỏ pHead = p_node , pTail = p_node (vì là phần tử đầu tiên của danh sách).
- Cho con trỏ pNext của pHead trỏ đến nullptr (đề phòng pHead của p_node không trỏ đến nullptr).
- Cho con trỏ pPrev của pHead trỏ đến nullptr (đề phòng pPrev của p_node không trỏ đến nullptr).
- Trả về con trỏ trỏ đến danh sách vừa mới khởi tạo.

3. bool addHead(d_List* &L, int data):

-Mục đích của hàm:

- Thêm vào đầu danh sách liên kết 1 node có giá trị là data (truyền vào hàm con trỏ của danh sách liên kết đôi, giá trị của data).

-Ý tưởng viết hàm:

- Khai báo 1 con trỏ mới newnode bằng hàm createNode có giá trị là data.
- Nếu danh sách trống hay pHead bằng nullptr , cho pHead của danh sách liên kết đôi bằng newnode và pTail của danh sách liên kết bằng newnode, trả về true.
- Nếu danh sách không trống thì cho newnode trỏ đến pHead (Chèn newnode là node đầu tiên của danh sách) và cho pHead trỏ đến newnode để cập nhật vị trí pHead, trả về true.

4. bool addTail(d_List* &L, int data):

-Mục đích của hàm:

- Thêm vào cuối danh sách liên kết 1 node có giá trị là data (truyền vào hàm con trỏ của danh sách liên kết đôi, giá trị của data).

-Ý tưởng của viết hàm:

- Khai báo 1 con trỏ mới newnode bằng hàm createNode có giá trị là data.
- Nếu danh sách trống hay pHead bằng nullptr , cho pHead của danh sách liên kết đôi bằng newnode và pTail của danh sách liên kết bằng newnode, trả về true.
- Nếu danh sách không trống thì cho pNext của pTail trỏ đến newnode (chèn newnode vào cuối danh sách) , cho pPrev của newNode bằng pTail , cập nhật vị tri pTail bằng newnode, trả về true.

5. bool removeHead(d_List* &L):

-Mục đích của hàm:

- Xóa node đầu của danh sách liên kết đôi (truyền vào hàm con trỏ của danh sách liên kết đôi).

-Ý tưởng viết hàm:

- Trường hợp pHead của danh sách đôi bằng nullptr(danh sách liên kết trống) thì trả về false.
- Trường hợp danh sách liên kết có 1 phần tử thì kiểm tra nếu con trỏ pHead của danh sách liên kết trỏ đến nullptr thì
  - Ta cho cho p_tail của danh sách liên kết bằng nullptr.
  - Khai báo 1 con trỏ temp có kiểu dữ liệu d_NODE bằng pHead , cho pHead bằng nullptr và cho pTail bằng nullptr, xóa node temp vừa khai báo.
- Trường hợp còn lại danh sách có ít nhất 2 phần tử thì:
  - Ta khai báo 1 con trỏ temp có kiểu dữ liệu d_NODE bằng pHead , cho pHead bằng node tiếp theo (chính là pNext của pHead)
  - Ở trước node đầu tiên là rỗng nên cho pPrev của pHead bằng nullptr sau đó ta xóa node temp vừa khai báo.
- Trả về true.

6. void removeTail(d_List *&L)

-Mục đích của hàm:

- Xóa node cuối của danh sách liên kết đôi(truyền vào hàm con trỏ của danh sách liên kết đôi).

-Ý tưởng viết hàm:

- Nếu pHead của danh sách bằng nullptr(danh sách liên kết trống) thì trả về false.
- Nếu danh sách liên kết có 1 phần tử thì kiểm tra, nếu con trỏ pHead của danh sách liên kết trỏ đến nullptr hay (pNext của pHead bằng nullptr) thì ta khai báo 1 con trỏ temp có kiểu dữ liệu d_NODE bằng pHead, cho pHead và pTail bằng nullptr, xóa temp và thoát khỏi hàm.
- Trường hợp còn lại (danh sách có 2 node trở lên):
  - Khai báo 1 con trỏ temp có kiểu dữ liệu d_NODE bằng pTail, cho pTail bằng Node đứng trước nó hay (pTail = pPrev của Ptail)
  - Cho pNext của pTail trỏ đến nullptr (do node cuối danh sách trỏ đến nullptr), xóa con trỏ temp.

7. void removeAll(d_List* &L)

-Mục đích của hàm:

- Xóa tất cả node có trong danh sách liên kết đôi (truyền vào hàm con trỏ của danh sách liên kết đôi).

-Ý tưởng viết hàm:

- Nếu pHead bằng nullptr (danh sách liên kết trống) thoát khỏi hàm.
- Khai báo 1 con trỏ cur có kiểu dữ liệu NODE bằng pTail.
- Duyệt qua từng node(xóa từng node): (xóa bắt đầu từ node cuối đi về node đầu)
  - Khai báo 1 con trỏ temp có kiểu dữ liệu NODE bằng cur.
  - Cho cur bằng cur->pPrev. (Chạy ngược về node trước)
  - Xóa temp.
- Cho pHead và pTail bằng nullptr.

8. removeBefore(d_List *&L, int val)

-Mục đích của hàm:

- Xóa 1 node nằm phía trước 1 node có giá trị là val (truyền vào hàm con trỏ của danh sách liên kết đôi, giá trị của val).

-Ý tưởng viết hàm:

- Trường hợp danh sách rỗng hoặc danh sách chỉ có 1 node hoặc giá trị của pHead bằng val thì thoát khỏi hàm (pHead hoặc pHead->pNext bằng nullptr hoặc key của pHead bằng val).
- Trường hợp có từ 2 node trở lên:
  - Khai báo node cur bằng pTail để duyệt qua các node (bắt đầu từ cuối danh sách).
  - Cho chạy vòng lặp cập nhật vị trí cur dần về phía trước danh sách liên kết:
    +Nếu tìm thấy Node có giá trị là val thì ta thoát khỏi vòng lặp.(đã tìm thấy node có giá trị bằng val).

- Sau vòng lặp:

  +Kiểm tra cur đã chạy đến cuối chưa, nếu cur bằng nullptr thì thoát khỏi hàm(không tìm được node nào có giá trị là val).

  +Khai báo node temp lưu giá trị node nằm phía trước node cần xóa (tức là temp bằng pPrev của cur).

  +Cập nhật vị trí phía sau của node cur (pPrev của cur = pPrev của temp).

  +Cập nhật vị trí phía trước của node nằm trước node cần xóa (cur->pPrev->pNext = cur).

  +Nếu temp bằng pHead thì cập nhật vị trí pHead bằng cur.

  +Xóa temp (node cần xóa).

9. void romveAfter(d_List* &L, int val)

-Mục đích của hàm:

- Xóa 1 node nằm phía sau 1 node có giá trị là val (truyền vào hàm con trỏ của danh sách liên kết đôi, giá trị của val).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết rỗng hoặc danh sách gồm 1 node hoặc pTail có giá trị là val thì ta thoát khỏi hàm do sau pTail không gồm node nào cả. (pHead hoặc pHead->pNext bằng nullptr hoặc key của pTail bằng val).
- Trường hợp còn lại có 2 node trở lên:
  - Khai báo node cur bằng pPrev của pTail (bỏ qua trường hợp bằng pTail) để duyệt qua các node (bắt đầu từ cuối danh sách).
  - Cho chạy vòng lặp cập nhật vị trí cur dần về phía trước danh sách liên kết:
    +Nếu tìm thấy Node có giá trị là val thì ta thoát khỏi vòng lặp.(đã tìm thấy node có giá trị bằng val).
  - Sau vòng lặp:
    +Kiểm tra cur đã chạy đến cuối chưa, nếu cur bằng nullptr thì thoát khỏi hàm (không tìm được node nào có giá trị là val).

    +Khai báo node temp lưu giá trị node nằm sau trước node cần xóa (tức là temp bằng pNext của cur).

    +Cập nhật vị trí nằm phía sau node có giá trị là val (pNext của cur bằng pNext của temp).

    +Cập nhật vị trí pPrev của node nằm sau node cần xóa (nếu node cần xóa là pTail ta chỉ cần cập nhật vị trí pTail bằng cur, nếu temp không phải là node cuối thì cho temp->pNext->pPrev bằng cur)

    +Xóa temp (node cần xóa).

10. bool addPos(d_List* &L, int data, int pos)

-Mục đích của hàm:

- Thêm node có giá trị là data vào vị trí nhất định của danh sách liên kết đôi(truyền vào con trỏ của danh sách liên kết, giá trị của data, vị trí cần chèn pos).

-Ý tưởng của hàm:

- Trường hợp vị trí nhỏ hơn 0 trả về false không thể chèn do vị trí không phù hợp.
- Trường hợp vị trí bằng 0 chạy hàm addHead và trả về giá trị true.
- Các trường hợp còn lại:
  - Khai báo con trỏ newnode có giá trị data bằng hàm createNode.
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Cho i bằng 0 chạy vòng lặp đến i bằng pos – 2, trong mỗi lần lặp nếu cur bằng nullptr thì xóa newnode và trả về false do vị trí chèn vượt quá số phần tử trong danh sách
  - Sau vòng lặp:

+Nếu cur bằng nullptr thì xóa newnode trả về false.(Khi cập nhật vị trí cur kiểm tra lại).

+Cho newnode trỏ đến node tiếp theo của cur hay pNext của newnode bằng pNext của cur (tức node tiếp theo cur là vị trí cần chèn).

+Cho pPrev của newnode bằng cur (cập nhật node phía sau của newnode)

+Cho cur trỏ đến newnode hay pNext của cur bằng newnode.

+Nếu newnode là phần tử cuối hay newnode trỏ đến nullptr thì cập nhật vị trí p_tail, nếu không phải là phần tử cuối ta cần cập nhật nối node phía sau của newnode với newnode (newnode->pNext->pPrev = newnode).

+Trả về true.

11. removePos(d_List *&L, int data, int pos)

-Mục đích của hàm:

- Xóa bỏ phần tử có vị trí pos của danh sách liên kết đôi tính từ vị trí 0 (truyền vào con trỏ của danh sách liên kết, giá trị của data, vị trí cần chèn pos).

-Ý tưởng của hàm:

- Trường hợp hàm là danh sách rỗng hoặc vị trí chèn nhỏ hơn 0 thì thoát khỏi hàm.
- Trường hợp vị trí chèn là 0 thì chạy hàm removeHead.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Cho i bằng 0 chạy vòng lặp đến i bằng pos – 2, trong mỗi lần lặp nếu cur bằng nullptr thì thoát khỏi hàm do vị trí chèn vượt quá số phần tử trong danh sách
  - Sau vòng lặp:

+Nếu cur bằng nullptr thì xóa newnode thoát khỏi hàm.(Khi cập nhật vị trí cur kiểm tra lại).

+Cho con trỏ temp bằng node ở sau cur hay temp bằng pNext của cur (node temp là node cần xóa).

+Cập nhật vị trí nằm sau phía sau node cur , temp nằm sau cur là node cần xóa (Cho pNext của cur bằng pNext của temp hoặc cur->pNext->Next).

+Kiểm tra temp có là node cuối hay không, nếu có cập nhật vị trí pTail bằng cur, nếu không thì cho node nằm sau node temp nối với cur (temp->pNext->Prev = cur).

+Xóa temp (node cần xóa).

12. bool addBefore(d_List *&L, int data, int val)

-Mục đích của hàm:

- Chèn Node có giá trị là data vào trước node có giá trị là val (nếu có) (truyền vào con trỏ của danh sách liên kết đôi, giá trị của data, giá trị của val).

-Ý tưởng viết hàm:

- Trường hợp hàm là danh sách rỗng thì thoát trả về giá trị false(không chèn được).
- Trường hợp node đầu của danh sách liên kết có giá trị bằng val thì chạy hàm addHead với node cần thêm có giá trị là data, trả về true.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
  - Chạy vòng lặp để xác định Node đứng trước Node có giá trị bằng val, trong quá trình chạy nếu tìm được Node thỏa thì thoát khỏi vòng lặp.
  - Sau vòng lặp:

+Kiểm tra xem đã chạy đến phần tử cuối chưa nếu đã chạy đến phần tử cuối tức là không có node nào có giá trị bằng val nên trả về false.

+Khai báo con trỏ của node mới newnode có giá trị là data.

+Để chèn newnode trước Node có giá trị bằng val tức là cur thì:

+Cho pPrev của newnode bằng pPrev của cur (cập nhật con trỏ trước của newnode).

+Cho cur->pPrev->pNext = newnode (node nằm trước node cần chèn nối với newnode).

+pPrev của cur bằng newnode (Cập nhật con trỏ trước của cur).

+Nối newnode với cur (newnode->pNext = cur) và trả về true.

13. bool addAfter(List* &L, int data, int val)

-Mục đích của hàm:

- Chèn Node có giá trị là data vào sau node có giá trị là val (nếu có) (truyền vào con trỏ của danh sách liên kết đôi, giá trị của data, giá trị của val).

-Ý tưởng viết hàm:

- Trường hợp hàm là danh sách rỗng thì thoát trả về giá trị false(không chèn được).
- Trường hợp node cuối có giá trị là val thì thêm node vào cuối danh sách qua hàm addTail.
- Các trường hợp còn lại:

- Khai báo con trỏ cur bằng p_Head để duyệt phần tử.
- Chạy vòng lặp xác định Node có giá trị bằng val, trong quá trình chạy nếu tìm được Node thỏa mãn thì thoát ra khỏi vòng lặp.
- Sau vòng lặp:

+Kiểm tra cur có bằng nullptr hay khồn nếu có tức là không có node nào có giá trị bằng val nên trả về false.

+Khai báo con trỏ của node mới newnode có giá trị là data.

+Để chèn newnode sau Node có giá trị bằng val tức là cur thì:

+Cho pPrev của newnode bằng cur (cập nhật con trỏ nằm ở phía trước newnode).

+Cho pNext của newnode bằng pNext của cur (cập nhật con trỏ nằm sau của newnode).

+Cho pPrev của pNext (cur) bằng newnode (cập nhật con trỏ nằm sau của node nằm sau node cur).

+pNext của cur bằng newnode (cập nhật con trỏ trước của cur) và trả về true.

14.void printList(d_List *L)

-Mục đích của hàm:

- In ra giá trị của từng Node có trong danh sách liên kết đôi (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết không gồm Node nào in ra NONE.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng pHead.
  - Duyệt qua lần lượt các Node có trong danh sách liên kết và in giá trị.

15. int countElements(d_List* L)

-Mục đích của hàm:

- Đếm số lượng Node , phần tử có trong danh sách liên kết (truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết không gồm Node nào trả về 0.
- Các trường hợp còn lại:
  - Khai báo con trỏ cur bằng pTail.
  - Khai báo biến đếm bằng 0.
  - Duyệt qua lần lượt các Node theo thứ tự ngược từ cuối danh sách liên kết đến đầu có trong danh sách liên kết và tăng giá trị biến đếm lên 1.
  - Trả về giá trị đếm sau khi hết vòng lặp.

16. d_List* reverseList(d_List* L)

-Mục đích của hàm:

- Tạo ra một danh sách liên kết đôi mới có dữ liệu ngược với danh sách liên kết đôi ban đầu(truyền vào hàm con trỏ của danh sách liên kết đôi).

-Ý tưởng viết hàm:

- Trường hợp danh sách rỗng thì ta trả về 1 danh sách rỗng.
- Các trường hợp còn lại:

Khai báo 1 node mới là node ban đầu có giá trị bằng giá trị node cuối của danh sách liên kết ban đầu

- Khai báo 1 danh sách liên kết mới có node đầu tiên là node vừa khai báo(Là danh sách liên kết đảo của danh sách ban đầu).
- Khai báo con trỏ cur bằng pPrev của pTail (node nằm trước pTail do đã lưu giá trị của pTail) của danh sách liên kết ban đầu (dùng để duyệt qua các node)
- Chạy vòng lặp để duyệt từng node: (chạy ngược từ cuối danh sách)

+Với mỗi vòng lặp ta khai báo 1 node mới newnode có giá trị là data của node đang xét.

+Thêm node mới newnode vào đuôi của danh sách liên kết đảo.

+Cho pNext của pTail bằng newnode.

+pPrev của newnode bằng pTail (cập nhật con trỏ trước).

+pTail bằng newnode(cập nhật vị trí cuối của node).

- Sau vòng lặp duyệt qua các phần tử thì trả về danh sách liên kết đảo.

17. void removeDuplicate(d_List *&L)

-Mục đích của hàm:

- Loại bỏ các Node có giá trị trùng trong danh sách liên kết đôi(truyền vào hàm con trỏ của danh sách liên kết).

-Ý tưởng viết hàm:

- Trường hợp danh sách liên kết trống hoặc danh sách liên kết có 1 phần tử thì thoát ra khỏi hàm.
- Khai báo con trỏ cur bằng pHead để duyệt qua các phần tử.
- Chạy vòng lặp cho cur đi qua từng node cho đến khi cur là phần tử cuối cùng:
  - Khai báo biến check kiểu bool bằng true (đảm bảo tính đúng đắn trong lúc duyệt).
  - Tiếp tục chạy vòng lặp khai báo con trỏ cur1 bằng cur chạy đến nullptr:

+Trong lúc chạy vòng lặp nếu giá trị của cur1 bằng giá trị của cur thì tức là Node đang xét (là Node đứng sau cur) đã bị trùng giá trị với 1 trong những Node đứng trước nó

+Khi bị trùng ta cho check bằng false, khai báo 1 con trỏ t bằng Node ta đang xét (Node đứng sau cur) .

+Nếu node cần xóa là node cuối thì ta cập nhật vị trí pTail bằng cur , nếu không phải là node cuối ta cập nhật con trỏ pPrev của node nằm sau t (t->pNext->pPrev = cur).

+Nối pNext của cur với pNext của t (Nối 2 node bỏ qua t, vì t là node cần xóa).

+Xóa t và thoát khỏi vòng lặp duyệt trùng.

- Sau vòng lặp nếu check bằng true thì ta xét đến node tiếp theo.

18. bool removeElement(d_List* &L, int key)

-Mục đích của hàm:

- Loai bỏ tất cả Node có giá trị là key trong danh sách liên kết đôi (truyền vào hàm con trỏ của danh sách liên kết đôi).

-Ý tưởng viết hàm:

- Trường hợp danh sách rỗng (pHead = nullptr) thì không có gì để xóa trả về false;
- Khai báo biến remove kiểu bool bằng false, sẽ kiểm tra có xóa phần tử nào không.
- Đầu tiên ta lặp liên tục để xóa khi phần tử đầu tiên có giá trị là key: (với điều kiện giá trị của pHead bằng key và pHead khác nullptr).
  - Khai báo con trỏ t bằng pHead, cập nhật vị trí pHead bằng node tiếp theo, cho remove bằng true khi xóa được ít nhất 1 phần tử.
  - Kiểm tra nếu sau pHead không là nullptr thì ta cập nhật con trỏ trước của pHead (pPrev của pHead) sau đó xóa con trỏ t.
- Kiểm tra khi xóa các phần tử đầu nếu danh sach rỗng cập nhật pTail bằng nullptr rồi trả về kết quả remove.
- Khai báo con trỏ cur bằng pHead để duyệt danh sách, bắt đầu duyệt từ phần tử thứ hai trở đi (điều kiện: cur->p_next != nullptr) để kiểm tra xem có node nào cần xóa nữa không:
  - Nếu node tiếp của cur có giá trị bằng key thì ta cần xóa node đó, cập nhật cur->p_next để bỏ qua node cần xóa, nếu node cần xóa là node cuối thì ta cập nhật vị trí p_tail bằng cur , nếu node cuối không là node cuối thì ta cập nhật con trỏ pPrev của node nằm sau node node cần xóa (pPrev của node nằm sau node cần xóa bằng cur), sau đó ta xóa node và cho remove bằng true.
  - Ngược lại nếu node tiếp không bị xóa thì ta cập nhật cur để duyệt (cur = cur->p_next) do nếu bị xóa ta vẫn cập nhật thì sẽ bỏ qua 1 Node vì cur->p_next đã thay đổi và ta cần kiểm tra lại.
- Cuối cùng sau vòng lặp ta trả về remove.

ẢNH UP CODE LÊN GITHUB: