課題：I111 5th Report　　言語：C++(Console Application)

氏名：GAO, Yuwei

学生番号：s1910092　　提出日：2019/05/09

① P10 のプログラムを元に末尾を表す tail を加えてみよう

79-93 行。Tail を通じて後ろにデータを追加メソッドも作成した

② tail があることによる，メリットデメリットをまとめよ．

メリット：

後ろにデータを追加しやすい、O(1)で。

デメリット：

if (tail == nullptr)tail = r;のような完璧ではないものが必要。

データを削除する際に削除対象が tail であるかとの判断も必要になる。

③ 双方向連結リストにしてみよう

8-14、121-144．データを前と後ろ両方に追加する方法を実装した。

④ 双方向連結リストのメリットデメリットをまとめよ。

メリット：

逆アクセスが便利。

どこでも前に戻れるので、使いやすい。

デメリット：

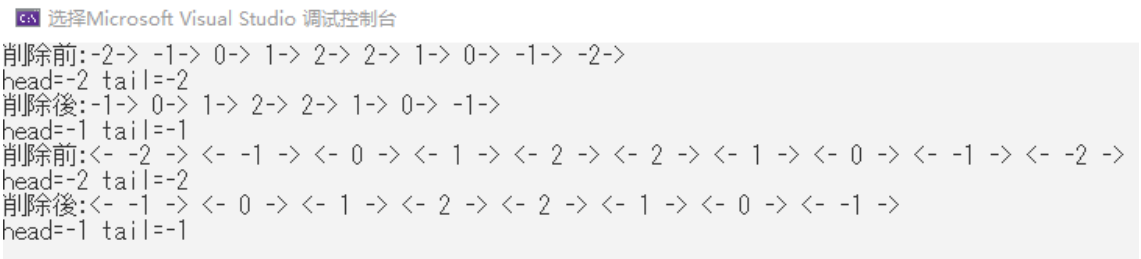
とある方面一方向連結リストよりやや重い、

例えばメモリ（CPU が miss cache 発生しやすくなる)、insert/delete する計算量。

⑤ ある数字を探索し，それを削除する関数 delete を追加してみよう

15-77．双方向連結リストと一方向連結リストの delete 関数を実装した。

そして両端の数字を検索し、削除した。



```cpp
#include <iostream>
class Node {
public:
    int data;
    Node* next;
    Node(int i, Node* n) : data(i), next(n) {}
};
class DoublyNode {//双方向連結リスト
public:
    int data;
    DoublyNode* pre;
    DoublyNode* next;
    DoublyNode(DoublyNode* b, int i, DoublyNode* n) :pre(b), data(i), next(n) {}
};
void delete_(Node* n, Node*& head, Node*& tail)//delete を追加
{
    if (n != tail)//tailではないなら
    {
        Node* del = n->next;
        n->data = del->data;
        n->next = del->next;
        if (del->next == nullptr)tail = n;
        delete(del);
    }
    else
    {
        if (n != head)//tailだけど、headではない
        {
            auto i = head;
            while (i->next != n)
            {
                i = i->next;
            }
            i->next = nullptr;
            tail = i;
            delete(n);
        }
        else//tail&&head
        {
            head = tail = nullptr;
            delete(n);
        }
    }
}
void delete_(DoublyNode* n, DoublyNode*& head, DoublyNode*& tail)// delete を追加
{
    if (n != tail)//tailではないなら
    {
        DoublyNode* del = n->next;
        n->data = del->data;
        n->next = del->next;
        if (del->next == nullptr)//次のやつもtailではないなら
        {
            tail = n;
```

```
55              delete(del);
56          }
57          else
58          {
59              del->next->pre = n;
60              delete(del);
61          }
62      }
63      else
64      {
65          if (n != head)//tailだけど、headではない
66          {
67              tail = n->pre;
68              tail->next = nullptr;
69              delete(n);
70          }
71          else//tail&&head
72          {
73              head = tail = nullptr;
74              delete(n);
75          }
76      }
77  }
78  void headAndTail() {
79      Node* head = nullptr;
80      Node* tail = nullptr;//末尾を表すtailを加えて
81      int array_[]{ 2,1,0,-1,-2 };
82      for (int& i : array_)//先頭にinsert
83      {
84          Node* r = new Node(i, head);
85          head = r;
86          if (tail == nullptr)tail = r;//今tail == nullptrなので、これを実行しな
              きゃ・・
87      }
88      for (int& i : array_)//後ろににinsert
89      {
90          Node* r = new Node(i, nullptr);
91          tail->next = r;
92          tail = r;
93      }
94      Node* n = head;
95      std::cout << "削除前:";
96      while (n != nullptr)//削除前状態を出力
97      {
98          std::cout << n->data << "-> ";
99          n = n->next;
100     }
101     std::cout << "\nhead=" << head->data << " tail=" << tail->data << "\n";
102     n = head;
103     while (n != nullptr)//ある数字を探索し，-2を削除する関数
104     {
105         if (n->data == -2)
106         {
107             delete_(n, head, tail);
108             n = head;
109         }
```

```
110         n = n->next;
111     }
112     n = head;
113     std::cout << "削除後:";
114     while (n != nullptr)
115     {
116         std::cout << n->data << "-> ";
117         n = n->next;
118     }
119     std::cout << "\nhead=" << head->data << " tail=" << tail->data << "\n";
120 }
121 void doubly() {//双方向連結リストを表す
122     DoublyNode* head = nullptr;
123     DoublyNode* tail = nullptr;
124     int array_[]{ 2,1,0,-1,-2 };
125     for (int& i : array_)//先頭にinsert
126     {
127         if (tail == nullptr || head == nullptr)//単に(tail == nullptr)を判断し ↵
              ても大丈夫
128         {
129             DoublyNode* r = new DoublyNode(nullptr, i, nullptr);
130             head = tail = r;
131         }
132         else
133         {
134             DoublyNode* r = new DoublyNode(nullptr, i, head);
135             head->pre = r;
136             head = r;
137         }
138     }
139     for (int& i : array_)//後ろににinsert
140     {
141         DoublyNode* r = new DoublyNode(tail, i, nullptr);
142         tail->next = r;
143         tail = r;
144     }
145     DoublyNode* n = head;
146     std::cout << "削除前:";
147     while (n != nullptr)//削除前状態を出力
148     {
149         std::cout << "<- " << n->data << " -> ";
150         n = n->next;
151     }
152     std::cout << "\nhead=" << head->data << " tail=" << tail->data << "\n";
153     n = head;
154     while (n != nullptr)//ある数字を探索し，-2を削除する関数
155     {
156         if (n->data == -2)
157         {
158             delete_(n, head, tail);
159             n = head;
160         }
161         n = n->next;
162     }
163     n = head;
164     std::cout << "削除後:";
```

```
165     while (n != nullptr)
166     {
167         std::cout << "<- " << n->data << " -> ";
168         n = n->next;
169     }
170     std::cout << "\nhead=" << head->data << " tail=" << tail->data << "\n";
171 }
172 int main()
173 {
174     headAndTail();
175     doubly();
176     return 0;
177 }
178 /*////////////実行結果/////////////////
179 削除前:-2-> -1-> 0-> 1-> 2-> 2-> 1-> 0-> -1-> -2->
180 head=-2 tail=-2
181 削除後:-1-> 0-> 1-> 2-> 2-> 1-> 0-> -1->
182 head=-1 tail=-1
183 削除前:<- -2 -> <- -1 -> <- 0 -> <- 1 -> <- 2 -> <- 2 -> <- 1 -> <- 0 -> <- -1 ↵
       -> <- -2 ->
184 head=-2 tail=-2
185 削除後:<- -1 -> <- 0 -> <- 1 -> <- 2 -> <- 2 -> <- 1 -> <- 0 -> <- -1 ->
186 head=-1 tail=-1
187 *////////////////////////////////////
```