データ構造とアルゴリズム実験レポート
課題：連結リスト, スタック, キュー

202110796 4 クラス 高橋大粋

締切日:2024 年 10 月 28 日
2024 年 10 月 28 日

1　基本課題

この課題では、教科書リスト 2.2,2.3(p.24),2.4,2.5(p.25) の連結リストの C プログラム linked_list.c, 教科書リスト 2.9,2.10 (p.32,33) のキューの C プログラム queue.c のリストおよび実行結果を示した。

1.1　連結リストの実装

1.1.1　実装の方針

まず、連結リストの機能としてセル p の直後に新しいセルを挿入する insert_cell(), リストの先頭に新しいセルを挿入する insert_cell_top(), セル p の直後のセルを削除する delete_cell(), リスト先頭のセルを削除する delete_cell_top(), リストの要素を順に標準出力する display() を linked_list.c に実装した。また、main 関数は別ファイル main_linked_list.c に実装した。

1.1.2　実装コードおよびコードの説明

プログラム 1 に、linked_list.c の主要部を示す。

void insert_cell(Cell *p, int d) 関数は入力としてセルへのポインタ p と int 型のデータ d を受け取り、新しいセル new_cell を作成し、new_cell のデータ部に d を、ポインタ部に p が指すポインタ部を代入する。そして p が指すセルのポインタ部に new_cell のアドレスを代入することでセル p の直後に新しいセルを挿入する。2 行目では Cell サイズ分のメモリ領域を動的に確保し、そのアドレスをポインタ new_cell に格納している。void insert_cell_top(int d) 関数は insert_cell() 関数と内容は殆ど同じだが、「p が指すセルのポインタ部」を先頭セルのアドレスである head に置き換えることでリストの先頭に新しいセルを挿入する。void delete_cell(Cell *p) 関数は入力としてセルへのポインタ p を受け取り、セルへのポインタ q に p が指すセルのポインタ部を代入し、p が指すセルのポインタ部に q が指すセルのポインタ部を代入し、q が指すセルが使用している主記憶領域を開放することで、セル p の直後のセルを削除する。void delete_cell_top(void) 関数はセルへのポインタ q に先頭セルへのポインタ head を代入して、head に q が指すセルのポインタ部を代入し、q が指すセルが使用している主記憶領域を開放することでリストの先頭のセルを削除する。void display(void) 関数はセルへのポインタ current に先頭セルへのポインタ head を代入して、current が指すデータ部を printf() 関数でコマンドラインに出力する。そして current に current が指すセルのポインタ部を

代入することで直後のセルへ移動する。これを current が NULL になるまで繰り返し、改行する。

```
1  void insert_cell(Cell *p, int d) {
2      Cell *new_cell = (Cell*)malloc(sizeof(Cell));
3      new_cell->data = d;
4      new_cell->next = p->next;
5      p->next = new_cell;
6  }
7
8  void insert_cell_top(int d) {
9    Cell *new_cell = (Cell*)malloc(sizeof(Cell));
10   new_cell->data = d;
11   new_cell->next = head;
12   head = new_cell;
13 }
14
15 void delete_cell(Cell *p) {
16     Cell *q = p->next;
17     p->next = q->next;
18     free(q);
19 }
20
21 void delete_cell_top(void) {
22     Cell *q = head;
23     head = q->next;
24     free(q);
25 }
26
27 void display(void) {
28     Cell *current = head;
29     while(current != NULL){
30         printf("%d ", current->data);
31         current = current->next;
32     }
33     printf("\n");
34 }
```

プログラム 1　linked_list.c の主要部

1.1.3　実行結果

まず、linked_list.c を以下の make コマンドを実行してコンパイルし、プログラムを実行する。

```
PS C:\Users\daiki\Desktop\DSA\kadai2> make
gcc   linked_list.o main_linked_list.o   -o linked_list
PS C:\Users\daiki\Desktop\DSA\kadai2> ./linked_list
1 2 3
1 3
3
```

main 関数の主要部は以下の通りである。

```
int main(void) {
    insert_cell_top(1);
    insert_cell(head, 3);
    insert_cell(head, 2);
    display();

    delete_cell(head);
    display();

    delete_cell_top();
    display();

    return EXIT_SUCCESS;
    }
```

まず、insert_cell_top(1) でリストの先頭にデータ部が 1 の新しいセルを挿入した。insert_cell(head, 3) で先頭セルの直後にデータ部が 3 の新しいセルを挿入した。同様にデータ部が 2 の新しいセルを挿入し、display() でリストの要素を標準出力した。出力の 1 行目に 1 2 3 と表示されているため、セルをリストの先頭に挿入できること、セルを指定したセルの直後に挿入できること、という要件を満たすことが確認できた。次に、delete_cell(head) で head の直後のセルを削除し、display() でリストの要素を標準出力した。出力の 2 行目に 1 3 と表示されているため、先頭セルの直後の 2 が削除されていることが確認できた。そして、delete_cell_top() でリストの先頭セルを削除し、display() でリストの要素を標準出力した。出力の 3 行目に 3 と表示されているため、先頭セルの 1 が削除されていることが確認できた。以上の処理は 1.1.2 で説明したように動作している。よって、セルをリストの先頭に挿入できること、セルを指定したセルの直後に挿入できること、先頭セルを削除できること、指定したセルの直後のセルを削除できること、という要件をすべて満たすことを確認できた。

1.2 キューの実装

1.2.1 実装の方針

まず、リングバッファのキューの機能としてサイズ len のキューを作成・初期化し、処理が完了したら Queue created!を標準出力し、Queue 型のポインタを返す create_queue(), キューに整数を格納する

enqueue(), キューから整数を取り出す dequeue(), キューの要素を先頭から順に出力する display(), キューを破棄し、処理が完了したら Queue deleted!を標準出力する delete_queue() を queue.c に実装した。また、main 関数は別ファイル main_queue.c に実装した。

1.2.2 実装コードおよびコードの説明

プログラム 2 に、queue.c の主要部を示す。

Queue *create_queue(int len) 関数は引数に int 型の len を受け取り、キュー q の length に len を、front と rear に-1 を代入して初期化する。そして printf で Queue created!を標準出力し、Queue 型のポインタ q を返す。void enqueue(Queue *q, int d) 関数は引数に Queue 型のポインタ q と int 型の d を受け取り、キューが一杯であるかを判定し、一杯であるならば標準エラーを出力する。そうでなければ front が-1 なら front に 0 を代入し、rear を +1 する。このときリングバッファであるのでキューのサイズ length で割った余りを rear に代入する。配列 buffer の rear の指す位置に整数 d を格納する。int dequeue(Queue *q) 関数は引数に Queue 型のポインタ q を取り、キューが空の時、標準エラーを出力し、そうでなければ int 型 data に front の指す位置のデータを代入し、front と rear が同じ値なら両方-1 を代入して初期化し、そうでなければリングバッファを考慮して front を +1 し、data の値を返す事により、キューから整数を取り出す。void display(Queue *q) は引数に Queue 型のポインタ q を取り、もしキューが空なら標準エラーを出し、そうでなければ front から rear までのキューの要素を printf で標準出力し、処理が終わったら改行する。void delete_queue(Queue *q) 関数は引数に Queue 型のポインタ q を取り、q が NULL でなければ q の配列と q の主記憶領域を開放し、printf で Queue deleted!を標準出力する。

```
1  Queue *create_queue(int len) {
2      Queue *q = (Queue *)malloc(sizeof(Queue));
3      if (q == NULL) {
4          exit(EXIT_FAILURE);
5          return NULL;
6      }
7      q->buffer = (int *)malloc(len * sizeof(int));
8      if (q->buffer == NULL) {
9          exit(EXIT_FAILURE);
10         free(q);
11         return NULL;
12     }
13     q->length = len;
14     q->front = -1;
15     q->rear = -1;
16     printf("Queue created!\n");
17     return q;
18 }
19
20 void enqueue(Queue *q, int d) {
```

```
21      if ((q->rear + 1) % q->length == q->front) {
22          fprintf(stderr, "Queue is full! Cannot enqueue %d.\n", d);
23          return;
24      }
25      if (q->front == -1) {
26          q->front = 0;
27      }
28      q->rear = (q->rear + 1) % q->length;
29      q->buffer[q->rear] = d;
30  }
31
32  int dequeue(Queue *q) {
33      if (q->front == -1) {
34          fprintf(stderr, "Queue is empty! Cannot dequeue.\n");
35          return -1;
36      }
37      int data = q->buffer[q->front];
38      if (q->front == q->rear) {
39          q->front = -1;
40          q->rear = -1;
41      } else {
42          q->front = (q->front + 1) % q->length;
43      }
44      return data;
45  }
46
47  void display(Queue *q) {
48      if (q->front == -1) {
49          fprintf(stderr, "Queue is empty! Nothing to display.\n");
50          return;
51      }
52      int i = q->front;
53      while (1) {
54          printf("%d ", q->buffer[i]);
55          if (i == q->rear) break;
56          i = (i + 1) % q->length;
57      }
58      printf("\n");
59  }
60
```

```c
void delete_queue(Queue *q) {
    if (q != NULL) {
        free(q->buffer);
        free(q);
        printf("Queue deleted!\n");
    }
}
```

プログラム 2　queue.c の主要部

1.2.3　実行結果

まず、queue.c を以下の make コマンドを実行してコンパイルし、プログラムを実行する。

```
PS C:\Users\daiki\Desktop\DSA\kadai2> make queue
gcc    -c -o queue.o queue.c
gcc   queue.o main_queue.o   -o queue
PS C:\Users\daiki\Desktop\DSA\kadai2> ./queue
Queue created!
== Test 1: Enqueue one element and dequeue it ==
10
Dequeued: 10
Queue is empty! Nothing to display.

== Test 2: Enqueue multiple elements and dequeue them in order ==
20 30 40
Dequeued: 20
Dequeued: 30
40

== Test 3: Wrap-around case ==
40 50 60 70 80
Dequeued: 40
50 60 70 80
50 60 70 80 90

== Test 4: Queue empty and full conditions ==
Queue is full! Cannot enqueue 100.
Dequeued: 50
Dequeued: 60
Dequeued: 70
Dequeued: 80
```

```
Dequeued: 90
Queue is empty! Nothing to display.
Attempt to dequeue from empty queue:
Queue is empty! Cannot dequeue.
Queue deleted!
```

main 関数の主要部は以下の通りである

```
int main(void) {
    Queue *test_q = create_queue(5);

    //1
    printf("== Test 1: Enqueue one element and dequeue it ==\n");
    enqueue(test_q, 10);
    display(test_q);
    printf("Dequeued: %d\n", dequeue(test_q));
    display(test_q);

    //2
    printf("\n== Test 2: Enqueue multiple elements and dequeue them in
        order ==\n");
    enqueue(test_q, 20);
    enqueue(test_q, 30);
    enqueue(test_q, 40);
    display(test_q);
    printf("Dequeued: %d\n", dequeue(test_q));
    printf("Dequeued: %d\n", dequeue(test_q));
    display(test_q);

    //3
    printf("\n== Test 3: Wrap-around case ==\n");
    enqueue(test_q, 50);
    enqueue(test_q, 60);
    enqueue(test_q, 70);
    enqueue(test_q, 80);
    display(test_q);

    printf("Dequeued: %d\n", dequeue(test_q));
    display(test_q);

    enqueue(test_q, 90);
```

```
33      display(test_q);
34
35      //4
36      printf("\n== Test 4: Queue empty and full conditions ==\n");
37
38      enqueue(test_q, 100);
39
40      while (test_q->front != -1) {
41          printf("Dequeued: %d\n", dequeue(test_q));
42      }
43      display(test_q);
44
45      printf("Attempt to dequeue from empty queue:\n");
46      dequeue(test_q);
47
48      delete_queue(test_q);
49
50      return EXIT_SUCCESS;
51  }
```

Test1 ではキューに整数を 1 つ格納し、それが取り出せていることを確認した。enqueue(q,10) と display(q) でキューに 10 を格納して表示し、dequeue(q) でキューから整数を取り出して printf() で出力した。Test2 では enqueue() でキューに整数を複数格納し、display() で表示させた後に、dequeue() で格納した順に取り出して printf() で出力した。Test3 では配列の末尾と先頭にまたがる場合をテストした。enqueue() で 5 つ値を格納してキューを一杯に満たした後、display() で表示させた。この状態で dequeue() で整数を取り出し、display() でキューの要素を表示させると 40 が取り出され要素が 4 つになっていることを確認した。その後、enqueue() で再度キューを満たし、display() で表示さ、要素が 5 つあることを確認した。Test4 ではキューが空の時の dequeue() とキューが満杯の時の enqueue() のエラーをテストした。現時点でキューが満杯なので enqueue() するとキューが満杯であり、enqueue できないというエラーが表示された。次に dequeue() でキューを空にした後で dequeue() をすると、キューが空で dequeue できないというエラーが表示されることを確認した。以上の処理は 1.2.2 で説明したように動作している。よって、課題で提示された要件をすべて満たすことを確認した。

2 発展課題

この課題では、ダミーの head を用いる双方向循環リストの機能を doublylinked_list.c の C プログラムのリストおよび実行結果を示した。

2.1 双方向循環リストの実装

2.1.1 実装の方針

まず、双方向循環リストの機能としてセルの生成および初期化を行い、Cell 型のポインタを返す関数 CreateCell(), this_cell の次にセル p を挿入する関数 InsertNext(Cell *this_cell,Cell p), this_cell の前にセル p を挿入する関数 InsertPrev(Cell *this_cell,Cell p), this_cell をリストから削除する関数 DeleteCell(Cell *this_cell), 与えられた整数 d を保持しているセルを this_cell から順に探し、見つかったらそのセルを返す。見つからなければ NULL を返す関数 SearchCell(Cell *this_cell,int d), リストの要素を this_cell から順に出力する関数 Display(Cell *this_cell) , 配列のデータを読み込みリストに追加する関数 ReadFromArray() , 配列にリストの要素を書き出す関数 WriteToArray(), ファイルからデータを読み込みリストに追加する関数 ReadFromFile() , ファイルにリストの要素を書き出す関数 WriteToFile() を doublylinked_list.c に実装した。また、main 関数は別ファイル main_doublylinked_list.c に実装した。

2.1.2 実装コードおよびコードの説明

プログラム 3 に、doublylinked_list.c の主要部を示す。

Cell *CreateCell(int d, bool is_head) は引数に int 型の d, 先頭かどうかを示す is_head を取り、malloc でメモリを確保し、確保に失敗した場合はプログラムを終了する。data フィールドに d を代入し、is_head フィールドに is_head の値を代入して先頭かどうかを指定する。セルが孤立状態になるように prev と next の両方のポインタを自身に設定する。is_head が false の場合は、リストに接続されていないことを示すエラーを表示し、最後に生成した新しいセルのポインタを返す。void InsertNext(Cell *this_cell, Cell *p) は引数として this_cell と挿入するポインタ p を取る。p の next を this_cell の next に、prev を this_cell に設定し、this_cell の次のセルの prev を p に、this_cell の next を p に更新することで、this_cell の次の位置にセル p を挿入する。void InsertPrev(Cell *this_cell, Cell *p) は引数に this_cell と p のポインタを取る。p の prev を this_cell の prev, next を this_cell に設定し、this_cell の前のセルの next を p に、this_cell の prev を p に更新することで、this_cell の前にセル p を挿入する。void DeleteCell(Cell *this_cell) は引数に this_cell を取る。this_cell が削除対象であれば、その前後のセルをお互いに接続するように設定を変更し、this_cell をリストから切り離す。最後に free で this_cell のメモリを解放する。Cell *SearchCell(Cell *this_cell, int d) は this_cell を起点にリストを巡回し、data フィールドが d と等しいセルを見つけると、そのセルのポインタを返す。見つからなかった場合には NULL を返す。this_cell が NULL の場合も NULL を返す。void Display(Cell *this_cell) は this_cell が NULL の場合は処理を終了する。それ以外の場合、リストの各セルを巡回し、各セルの data フィールドを標準出力に表示する。リストの終端に到達すると改行する。void ReadFromArray(Cell *head, int arr[], int size) は引数としてヘッドセル head と追加するデータを含む配列 arr, 配列の要素数 size を受け取る。配列内の各データに対し、新しいセルを CreateCell で作成し、InsertPrev を用いてヘッドセルの前に順に追加していく。void WriteToArray(Cell *head, int arr[], int *size) はリストの各要素を配列に順次コピーし、配列の長さを size に代入する。void ReadFromFile(Cell *head, const char *filename) は引数としてヘッドセル head, 読み込み対象のファイル名 filename を取る。ファイルを開き、整数データを読み込み、CreateCell と InsertPrev を使ってリストに追加する。読み込みが終わるとファイルを閉る。ファイルが開けなかった場合には標準エラー出力にエラーメッセージを出し、プログラムを終了する。void WriteToFile(Cell *head, const char *filename) はファイルを開き、リスト内の各セ

ルのデータを順にファイルに出力する。ファイルが開けない場合はエラーメッセージを出力し、プログラムを終了する。書き込みが完了したらファイルを閉じる。

```
1  Cell *CreateCell(int d, bool is_head) {
2      Cell *new_cell = (Cell *)malloc(sizeof(Cell));
3      if (new_cell == NULL) {
4          exit(EXIT_FAILURE);
5      }
6      new_cell->data = d;
7      new_cell->is_head = is_head;
8      new_cell->prev = new_cell;
9      new_cell->next = new_cell;
10
11     if (!is_head) {
12         printf("Warning: Cell with data %d is not connected to the list.\n"
               , d);
13     }
14     return new_cell;
15 }
16
17 void InsertNext(Cell *this_cell, Cell *p) {
18     if (this_cell == NULL || p == NULL) return;
19     p->next = this_cell->next;
20     p->prev = this_cell;
21     this_cell->next->prev = p;
22     this_cell->next = p;
23 }
24
25 void InsertPrev(Cell *this_cell, Cell *p) {
26     if (this_cell == NULL || p == NULL) return;
27     p->prev = this_cell->prev;
28     p->next = this_cell;
29     this_cell->prev->next = p;
30     this_cell->prev = p;
31 }
32
33 void DeleteCell(Cell *this_cell) {
34     if (this_cell == NULL || this_cell->is_head) return;
35     this_cell->prev->next = this_cell->next;
36     this_cell->next->prev = this_cell->prev;
```

```
37      free(this_cell);
38  }
39
40  Cell *SearchCell(Cell *this_cell, int d) {
41      if (this_cell == NULL) return NULL;
42
43      Cell *current = this_cell->next;
44      while (current != this_cell) {
45          if (current->data == d) {
46              return current;
47          }
48          current = current->next;
49      }
50      return NULL;
51  }
52
53  void Display(Cell *this_cell) {
54      if (this_cell == NULL) return;
55      Cell *current = this_cell->next;
56      while (!current->is_head) {
57          printf("%d ", current->data);
58          current = current->next;
59      }
60      printf("\n");
61  }
62
63  void ReadFromArray(Cell *head, int arr[], int size) {
64      for (int i = 0; i < size; i++) {
65          InsertPrev(head, CreateCell(arr[i], false));
66      }
67  }
68
69  void WriteToArray(Cell *head, int arr[], int *size) {
70      int i = 0;
71      Cell *current = head->next;
72      while (!current->is_head) {
73          arr[i++] = current->data;
74          current = current->next;
75      }
76      *size = i;
```

```
77  }
78
79  void ReadFromFile(Cell *head, const char *filename) {
80      FILE *file = fopen(filename, "r");
81      if (file == NULL) {
82          fprintf(stderr, "Failed to open file %s\n", filename);
83          exit(EXIT_FAILURE);
84      }
85      int value;
86      while (fscanf(file, "%d", &value) != EOF) {
87          InsertPrev(head, CreateCell(value, false));
88      }
89      fclose(file);
90  }
91
92  void WriteToFile(Cell *head, const char *filename) {
93      FILE *file = fopen(filename, "w");
94      if (file == NULL) {
95          fprintf(stderr, "Failed to open file %s\n", filename);
96          exit(EXIT_FAILURE);
97      }
98      Cell *current = head->next;
99      while (!current->is_head) {
100         fprintf(file, "%d ", current->data);
101         current = current->next;
102     }
103     fprintf(file, "\n");
104     fclose(file);
105 }
```

プログラム 3　doublylinked_list.c の主要部

2.1.3　実行結果

まず、doublylinked_list.c を以下の make コマンドを実行してコンパイルし、プログラムを実行する。

```
1  PS C:\Users\daiki\Desktop\DSA\kadai2> make doublylinked_list
2  gcc    -c -o main_doublylinked_list.o main_doublylinked_list.c
3  gcc   doublylinked_list.o main_doublylinked_list.o   -o doublylinked_list
4  PS C:\Users\daiki\Desktop\DSA\kadai2> ./doublylinked_list
5  Initial list created with head node.
6
```

```
== Inserting cells ==
Warning: Cell with data 1 is not connected to the list.
Warning: Cell with data 2 is not connected to the list.
Warning: Cell with data 3 is not connected to the list.
1 2 3

== Checking unconnected cell ==
Warning: Cell with data 4 is not connected to the list.
Warning: Cell with data 4 is not connected to the list.

== Deleting cells ==
2 3
2


== Reinserting and searching for cells ==
Warning: Cell with data 1 is not connected to the list.
Warning: Cell with data 2 is not connected to the list.
Warning: Cell with data 3 is not connected to the list.
1 2 3

Searching for cells:
Found cell with data 1 at address 000001f941cb1460
Found cell with data 2 at address 000001f941cb1480
Found cell with data 3 at address 000001f941cb14a0

List and all cells deleted.
```

main 関数の主要部は以下の通りである

```
int main(void) {
    Cell *head = CreateCell(0, true);
    printf("Initial list created with head node.\n");

    printf("\n== Inserting cells ==\n");
    Cell *first = CreateCell(1, false);
    InsertNext(head, first);

    Cell *middle = CreateCell(2, false);
    InsertNext(first, middle);

```

```
12      Cell *last = CreateCell(3, false);
13      InsertPrev(head, last);
14
15      Display(head);
16
17
18      printf("\n== Checking unconnected cell ==\n");
19      Cell *unconnected = CreateCell(4, false);
20      if (unconnected->next == unconnected && unconnected->prev ==
            unconnected) {
21          printf("Warning: Cell with data %d is not connected to the list.\n
                ", unconnected->data);
22      }
23
24      printf("\n== Deleting cells ==\n");
25      DeleteCell(first);
26      Display(head);
27
28      DeleteCell(last);
29      Display(head);
30
31      DeleteCell(middle);
32      Display(head);
33
34      printf("\n== Reinserting and searching for cells ==\n");
35      Cell *new_first = CreateCell(1, false);
36      Cell *new_middle = CreateCell(2, false);
37      Cell *new_last = CreateCell(3, false);
38
39      InsertNext(head, new_first);
40      InsertNext(new_first, new_middle);
41      InsertPrev(head, new_last);
42      Display(head);
43
44      printf("\nSearching for cells:\n");
45      Cell *found;
46      found = SearchCell(head, 1);
47      if (found != NULL) printf("Found cell with data 1 at address %p\n", (
            void *)found);
48      else printf("Cell with data 1 not found.\n");
```

```
49
50      found = SearchCell(head, 2);
51      if (found != NULL) printf("Found cell with data 2 at address %p\n", (
            void *)found);
52      else printf("Cell with data 2 not found.\n");
53
54      found = SearchCell(head, 3);
55      if (found != NULL) printf("Found cell with data 3 at address %p\n", (
            void *)found);
56      else printf("Cell with data 3 not found.\n");
57
58      DeleteCell(new_first);
59      DeleteCell(new_middle);
60      DeleteCell(new_last);
61
62      free(head);
63      printf("\nList and all cells deleted.\n");
64
65      return EXIT_SUCCESS;
66  }
```

first に先頭に挿入するセル、middle に中間に挿入するセル、last に末尾に挿入するセルを割り当て、InsertNext と InsertPrev で挿入する。この際、生成しただけで挿入していないセルがリストに繋がれていないことをメッセージとして出力している。Display で表示させると 1 2 3 が表示された。次にリストに接続されていないセルを作成し、メッセージを表示させた。Deleting cells では先頭セル、末尾セル、中間セルを削除できることを確認した。最後に新しく先頭、中間、末尾セルを作成し、SearchCell により先頭、中間、末尾セルを検索できることを確認した。

ReadFromArray, WriteToArray, WriteToFile の動作テストを以下の main 関数で行った。

```
1   void main(void){
2       Cell *head = CreateCell(0, true);
3       printf("Initial list created with head node.\n");
4
5       printf("\n== ReadFromArray() Test ==\n");
6       int data[] = {1, 2, 3, 4, 5};
7       int size = sizeof(data) / sizeof(data[0]);
8       ReadFromArray(head, data, size);
9       printf("List after reading from array: ");
10      Display(head);
11
12      printf("\n== WriteToArray() Test ==\n");
```

```
    int output[10];
    int output_size;
    WriteToArray(head, output, &output_size);
    printf("Array after writing from list: ");
    for (int i = 0; i < output_size; i++) {
        printf("%d ", output[i]);
    }
    printf("\n");

    printf("\n== WriteToFile() Test ==\n");
    const char *filename = "output.txt";
    WriteToFile(head, filename);
    printf("List data written to file '%s'\n", filename);

    printf("\n== ReadFromFile() Test ==\n");

    while (head->next != head) {
        DeleteCell(head->next);
    }
    printf("List cleared.\n");

    ReadFromFile(head, filename);
    printf("List after reading from file: ");
    Display(head);

    while (head->next != head) {
        DeleteCell(head->next);
    }
    free(head);
    printf("\nList and all cells deleted.\n");

    return EXIT_SUCCESS;
}
```

出力結果は以下の通り

```
PS C:\Users\daiki\Desktop\DSA\kadai2> ./doublylinked_list
Initial list created with head node.

== ReadFromArray() Test ==
Warning: Cell with data 1 is not connected to the list.
```

```
Warning: Cell with data 2 is not connected to the list.
Warning: Cell with data 3 is not connected to the list.
Warning: Cell with data 4 is not connected to the list.
Warning: Cell with data 5 is not connected to the list.
List after reading from array: 1 2 3 4 5

== WriteToArray() Test ==
Array after writing from list: 1 2 3 4 5

== WriteToFile() Test ==
List data written to file 'output.txt'

== ReadFromFile() Test ==
List cleared.
Warning: Cell with data 1 is not connected to the list.
Warning: Cell with data 2 is not connected to the list.
Warning: Cell with data 3 is not connected to the list.
Warning: Cell with data 4 is not connected to the list.
Warning: Cell with data 5 is not connected to the list.
List after reading from file: 1 2 3 4 5

List and all cells deleted.
```

例として配列 1,2,3,4,5 を ReadFromArray で読み込み、リストに追加して Dsplay で表示させた。1 2 3 4 5 と表示されていることを確認した。WriteToArray のテストではリストの要素を配列に書き出し、printf で配列の要素を表示させた。1 2 3 4 5 と表示されていることを確認した。WriteToFile のテストではリストのデータを output.txt に書き出した。ReadFromFile のテストではリストをクリアしてから、ファイルからデータを読み込み、リストに追加して Dsplay で表示させた。1 2 3 4 5 と表示されていることを確認した。以上の処理は 2.1.2 で説明したように動作している。よって課題の要件をすべて満たしていることを確認した。