キューの作成

22060 211 古城隆人

2024 年 7 月 25 日

1　目的

キューのデータ構造を理解し、実装することで、キューの基本的な動作を理解する。キューを実装する方法として配列、リングバッファ、リスト構造を用いて実装を行う。リスト構造ではメモリの確保と解放を行うので、メモリの確保と解放の方法の理解も目的とする。

2　原理

2.1　キュー

キューは、データを先入れ先出し (FIFO) で処理するデータ構造である。キューは、データを追加する enqueue とデータを取り出す dequeue の 2 つの操作を持つ。enqueue はキューの末尾にデータを追加し、dequeue はキューの先頭からデータを取り出す。キューは、配列、リングバッファ、リスト構造を用いて実装することができる。

2.2　配列

配列を用いてキューを実装する場合、配列の先頭をキューの先頭とし、末尾をキューの末尾とする。enqueue は末尾にデータを追加し、dequeue は先頭からデータを取り出す。でキューされるたびに配列の要素をずらすため、演算にかかる時間が長くなる。

2.3　リングバッファ

リングバッファを用いてキューを実装する場合、配列の先頭と末尾をつなげて演算を行う。また、キューの先頭と配列のインデックスの数を格納するための変数も用意する。この方法では、配列の要素をずらすことなくキューを実装することができる。データ構造を 2 に示す。

ソースコード 1　queue.h

```
1  #define QUEUE_SIZE 5
2  struct queue
3  {
4      int array[QUEUE_SIZE]; // データが入る配列
5      int wp;                // 次にデータを入れる場所の配列番号
6      int quantity;          // データの個数
7  };
```

2.4　リスト構造

リスト構造を用いてキューを実装する場合、メモリの確保を行ってデータを格納する。リスト構造は、データを格納するための構造体と次のデータを指すポインタを持つ。データ構造を 2 に示す。

ソースコード 2　queue.h

```
1  struct queue
```

```
2  {
3      int val;
4      struct queue *addr;
5  };
6  struct queue *bottom_queue = NULL; // 最古のキューのアドレスを記憶しておくポインタ
7  struct queue *top_queue = NULL;    // 最新のキューのアドレスを記憶しておくポインタ
```

プログラムのコメントアウトにもあるが、bottom_queue は最古のキューのアドレスを記憶しておくポインタであり、top_queue は最新のキューのアドレスを記憶しておくポインタである。

3　実験環境

実験環境を表 1 に示す。

項目	値
OS	windows10 上の wsl2(Ubuntu)
CPU	Intel Core i7 11800H
メモリ	8GB
コンパイラ	gcc 11.4.0

表 1　実験環境

4　プログラムの設計と説明

4.1　配列を用いたキューの実装

配列を用いてキューを作成するために関数の作成を行う。

4.1.1　作成した関数

・enqueue 関数

このプログラムでは、配列にキューを追加する。配列はグローバル変数と宣言しているため引数はキューに追加するでーただけである。

表 2　enqueue 関数

機能	キューにデータを追加する。
引数	int data：追加するデータ
戻り値	int：エラーコード (-100: 正常終了, -101: キューが満杯, -102: データが自然数でない)

ソースコード 3　enqueue 関数

```
1  int enqueue(int data)
2  {
```

```
3       // 残り領域があるか確認する
4       if (quantity >= QUEUE_SIZE)
5       {
6           return -101;
7       }
8       // データが自然数か確認する
9       if (data < 0)
10      {
11          return -102;
12      }
13      // 配列のキューにデータを保存する
14      queue[quantity] = data;
15      // キューのポインタをインクリメントする
16      quantity++;
17      // 返り値を返す
18      return -100;
19  }
```

・dequeue 関数

このプログラムでは、配列からキューを取り出す。配列はグローバル変数と宣言しているため引数はない。返り値として取り出したデータを返す。

表 3　dequeue 関数

機能	キューからデータを取り出す。
引数	なし
戻り値	int：エラーコード (-201: データが存在しない, データ: 正常終了)

ソースコード 4　dequeue 関数

```
1   int dequeue(void)
2   {
3       // データが存在するかどうか確認する.
4       if (quantity <= 0)
5       {
6           return -201;
7       }
8       // キューからデータをとりだす.
9       int data = queue[0];
10      // データの個数カウントを減らす
11      quantity--;
12      // 取り出されたデータ部分を埋めるように再構築する
13      for (int i = 0; i < quantity; i++)
14      {
15          queue[i] = queue[i + 1];
16      }
17      // 空き領域を初期化する.
18      queue[quantity] = 0;
```

```
19      // データもしくは返り値を返す
20      return data;
21  }
```

・initQueue 関数

このプログラムでは、配列を初期化する。配列はグローバル変数と宣言しているため引数はない。配列の中身をすべて 0 にすることで初期化を行う。

表 4　initQueue 関数

機能	キューを初期化する。
引数	なし
戻り値	int：エラーコード (0: 正常終了)

ソースコード 5　initQueue 関数

```
1   int initQueue()
2   {
3       // キューのデータを入れる配列をすべて 0 に初期化する.
4       for (int i = 0; i < QUEUE_SIZE; i++)
5       {
6           queue[i] = 0;
7       }
8       // 格納データ個数を 0 に初期化する.
9       quantity = 0;
10      return 0;
11  }
```

・showQueue 関数

このプログラムでは、配列の中身を表示する。配列はグローバル変数と宣言しているため引数はない。配列の中身をすべて表示する。

表 5　showQueue 関数

機能	キューの中身を表示する。
引数	なし
戻り値	int：エラーコード (0: 正常終了)

ソースコード 6　showQueue 関数

```
1   int showQueue()
2   {
3       // 配列全体のデータを順に表示する.
4       // データとデータの間に区切り文字「|」を表示する.
5       for (int i = 0; i < QUEUE_SIZE; i++)
6       {
7           printf("%d|", queue[i]);
8       }
```

```
 9      return 0;
10  }
```

・showResult 関数

このプログラムでは、エラーコードに応じてエラーメッセージを表示する。引数としてエラーコードを受け取り、エラーコードに応じたエラーメッセージを表示する。

表 6　showResult 関数

機能	関数の実行結果を表示する。
引数	int result : エラーコード
戻り値	なし

ソースコード 7　showResult 関数

```
 1    void showResult(int result)
 2  {
 3      // result の値に応じて，対応するエラーメッセージを表示する．
 4      showQueue();
 5      printf("--> %d:", result);
 6      switch (result)
 7      {
 8      case -100:
 9          printf("enqueue 成功\n");
10          break;
11      case -101:
12          printf("エラー：キューが満杯です\n");
13          break;
14      case -102:
15          printf("エラー：データが自然数ではありません\n");
16          break;
17      case -201:
18          printf("エラー：キューが空です\n");
19          break;
20      default:
21          if (result < 0)
22          {
23              printf("エラー：不明なエラーです\n");
24          }
25          else
26          {
27              printf("dequeue(%d)\n", result);
28          }
29          break;
30      }
31  }
```

・main 関数

キューが正しく動作しているかを確かめるために main 関数を作成する。main 関数はキューの初期化、

enqueue、dequeue、showQueue、showResult を行う。キューをあふれさせるために 5 回 enqueue を行っている。また、キューが空かどうかの判定ができているかを確かめるために 5 回 dequeue を行っている。グローバル変数として定義している配列は、インデックスが 4 個しかないため 5 回操作を行うとエラーが出てくるはずである。

ソースコード 8　main 関数

```
 1  int main()
 2  {
 3      // キューを初期化する
 4      initQueue();
 5      // キューにデータを追加する
 6      showResult(enqueue(1));
 7      showResult(enqueue(2));
 8      showResult(enqueue(3));
 9      showResult(enqueue(4));
10      showResult(enqueue(5));
11      // キューの中身を表示する
12      // showQueue();
13      // キューからデータを取り出す
14      showResult(dequeue());
15      showResult(dequeue());
16      showResult(dequeue());
17      showResult(dequeue());
18      showResult(dequeue());
19      // キューの中身を表示する
20      // showQueue();
21      return 0;
22  }
```

4.2　リングバッファを用いたキューの実装

リングバッファを用いてキューを作成するために関数の作成を行う。

4.2.1　作成した関数

・enqueue 関数

このプログラムでは、リングバッファにキューを追加する。リングバッファは構造体で宣言しているため引数は構造体と追加するデータである。

表 7　enqueue 関数

機能	キューにデータを追加する。
引数	struct queue *obj：キューの構造体, int data：追加するデータ
戻り値	int：エラーコード (-100: 正常終了, -101: キューが満杯, -102: データが自然数でない)

ソースコード 9　enqueue 関数

```
1  int enqueue(struct queue *obj, int data)
2  {
3      // 残り領域があるか確認する.
4      // データが自然数か確認する.
5      // 配列にデータを保存する.
6      // 最新のデータが入った場所の次の場所を指すように値を更新する.
7      // データの個数カウントを増やす.
8      // 返り値を返す.
9      if (obj->quantity >= QUEUE_SIZE)
10     {
11         return -101;
12     }
13     if (data < 0)
14     {
15         return -102;
16     }
17     obj->array[obj->wp] = data;
18     obj->wp = (obj->wp + 1) % QUEUE_SIZE;
19     obj->quantity++;
20     return -100;
21 }
```

・dequeue 関数

このプログラムでは、リングバッファからキューを取り出す。リングバッファは構造体で宣言しているため引数は構造体である。

表 8　dequeue 関数

機能	キューからデータを取り出す。
引数	struct queue *obj：キューの構造体
戻り値	int：エラーコード (-201: データが存在しない, データ: 正常終了)

ソースコード 10　dequeue 関数

```
1  int dequeue(struct queue *obj)
2  {
3      // データが存在するかどうか確認する.
4      // キューからデータをとりだす.
5      // データの個数のカウントを減らす.
6      // データもしくは返り値を返す.
7      if (obj->quantity <= 0)
8      {
9          return -201;
10     }
11     int data = obj->array[0];
12     obj->quantity--;
13     obj->array[(obj->wp+QUEUE_SIZE - obj->quantity - 1) % QUEUE_SIZE] = 0;
```

```
14      return data;
15  }
```

・initQueue 関数

このプログラムでは、リングバッファを初期化する。リングバッファは構造体で宣言しているため引数は構造体である。

表 9　initQueue 関数

機能	キューを初期化する。
引数	struct queue *obj：キューの構造体
戻り値	int：エラーコード (0: 正常終了)

ソースコード 11　initQueue 関数

```
1   int initqueue(struct queue *obj)
2   {
3       // キューのデータを入れる配列をすべて 0 に初期化する.
4       // キューのデータ格納個数を 0 に初期化する
5       // キューの wp ポインタを 0 に初期化する.
6       for (int i = 0; i < QUEUE_SIZE; i++)
7       {
8           obj->array[i] = 0;
9       }
10      obj->quantity = 0;
11      obj->wp = 0;
12      return 0;
13  }
```

・showQueue 関数

このプログラムでは、リングバッファの中身を表示する。リングバッファは構造体で宣言しているため引数は構造体である。

表 10　showQueue 関数

機能	キューの中身を表示する。
引数	struct queue *obj：キューの構造体
戻り値	int：エラーコード (0: 正常終了)

ソースコード 12　showQueue 関数

```
1   int showQueue(struct queue *obj)
2   {
3       // 配列全体のデータを順に表示する.
4       // データとデータの間に区切り文字「/」を表示する.
5       for (int i = 0; i < QUEUE_SIZE; i++)
6       {
7           printf("%d", obj->array[i]);
```

```
 8          if (i < QUEUE_SIZE - 1)
 9          {
10              printf("|");
11          }
12      }
13      return 0;
14  }
```

・showResult 関数

このプログラムでは、エラーコードに応じてエラーメッセージを表示する。引数としてエラーコードを受け取り、エラーコードに応じたエラーメッセージを表示する。

表 11　showResult 関数

機能	関数の実行結果を表示する。
引数	int result：エラーコード
戻り値	なし

ソースコード 13　showResult 関数

```
 1  void showResult(int result)
 2  {
 3      // result の値に応じて，対応するエラーメッセージを表示する．
 4      switch (result)
 5      {
 6      case -100:
 7          printf("正常終了\n");
 8          break;
 9      case -101:
10
11          printf("エラー：キューが満杯です\n");
12          break;
13      case -102:
14          printf("エラー：データが自然数ではありません\n");
15          break;
16      case -201:
17          printf("エラー：キューが空です\n");
18          break;
19      default:
20          printf("\n");
21          break;
22      }
23  }
```

・main 関数

キューが正しく動作しているかを確かめるために main 関数を作成する。main 関数はキューの初期化、enqueue、dequeue、showQueue、showResult を行う。キューをあふれさせるために 6 回 enqueue を行って

いる。また、キューが空かどうかの判定ができているかを確かめるために 6 回 dequeue を行っている。定義にて配列の数を 5 個としているため、6 回目の enqueue でエラーが出力されるはずである。

ソースコード 14　main 関数

```
#define QUEUE_SIZE 5
int main(void)
{
    struct queue obj;
    int result;
    result = enqueue(&obj,40);
    initqueue(&obj);
    showQueue(&obj);
    printf("<40");
    showResult(result);
    result = enqueue(&obj,60);
    showQueue(&obj);
    printf("<60");
    showResult(result);
    result = enqueue(&obj,10);
    showQueue(&obj);
    printf("<10");
    showResult(result);
    result = enqueue(&obj,80);
    showQueue(&obj);
    printf("<80");
    showResult(result);
    result = enqueue(&obj,30);
    showQueue(&obj);
    printf("<30");
    showResult(result);
    result = enqueue(&obj,50);
    showQueue(&obj);
    printf("<50");
    showResult(result);
    result = dequeue(&obj);
    showQueue(&obj);
    printf(">%d", result);
    showResult(result);
    result = enqueue(&obj,1);
    showQueue(&obj);
    printf("<1");
    showResult(result);
    result = enqueue(&obj,2);
    showQueue(&obj);
    printf("<2");
    showResult(result);
    result = dequeue(&obj);
    showQueue(&obj);
    printf(">%d", result);
```

```
46      showResult(result);
47      result = dequeue(&obj);
48      showQueue(&obj);
49      printf(">%d", result);
50      showResult(result);
51      result = dequeue(&obj);
52      showQueue(&obj);
53      printf(">%d", result);
54      showResult(result);
55      result = dequeue(&obj);
56      showQueue(&obj);
57      printf(">%d", result);
58      showResult(result);
59      result = dequeue(&obj);
60      showQueue(&obj);
61      printf(">%d", result);
62      showResult(result);
63      result = dequeue(&obj);
64      showQueue(&obj);
65      printf(">%d", result);
66      showResult(result);
67      result = dequeue(&obj);
68      showQueue(&obj);
69      printf(">%d", result);
70      showResult(result);
71      result = dequeue(&obj);
72      showQueue(&obj);
73      printf(">%d", result);
74      showResult(result);
75      result = enqueue(&obj,200);
76      showQueue(&obj);
77      printf("<200");
78      showResult(result);
79      result = enqueue(&obj,300);
80      showQueue(&obj);
81      printf("<300");
82      showResult(result);
83      result = enqueue(&obj,-400);
84      showQueue(&obj);
85      printf("<-400");
86      showResult(result);
87      result = enqueue(&obj,500);
88      showQueue(&obj);
89      printf("<500");
90      showResult(result);
91      return 0;
92  }
```

4.3　リスト構造を用いたキューの実装

リスト構造を用いてキューを作成するために関数の作成を行う。

4.3.1　作成した関数

・enqueue 関数

このプログラムでは、リスト構造にキューを追加する。構造体はグローバル変数で宣言しているため、引数は追加するデータのみである。

表 12　enqueue 関数

機能	キューにデータを追加する。
引数	int data：追加するデータ
戻り値	int：エラーコード (-100: 正常終了, -101: メモリの確保ができない, -102: データが自然数でない)

ソースコード 15　enqueue 関数

```
1  int enqueue(int data)
2  {
3      int r_val = -100;                          // 返り値を宣言する
4      struct queue *new_queue;                   // 新しく確保する領域（構造体）へのポイン
         タを宣言する
5      if (data < 0 /*挿入するデータが 0 以下であったとき*/) // 0 以下のデータが入力された時
6      {
7          r_val = -102; // 返り値を設定する
8      }
9      // 追加する領域を確保する
10     /*malloc で構造体のオブジェクトの領域を確保して，確保失敗した時*/
11     else if ((new_queue = (struct queue *)malloc(sizeof(struct queue))) == NULL)
12     {
13         // 確保できなかったとき
14         r_val = -101;
15     }
16     else
17     {
18         // 新しいキューのデータ値として引数のデータを入れる(enqueue:2-1)
19         new_queue->val = data;
20         // 新しいキューのアドレス値として NULL を入れる(enqueue:2-2)
21         new_queue->addr = NULL;
22         // まだ 1 個のキューもない時
23         if (bottom_queue == NULL)
24         {
25             // 新しいキューが最古であるので，最古のキューを指すアドレスを更新する(enqueue
               :4-1)
26             bottom_queue = new_queue;
```

```
27              // 新しいキューが最新であるので，最新のキューを指すアドレスを更新する(enqueue
                  :4-2)
28              top_queue = new_queue;
29          }
30          // 1 個以上のキューがあるとき
31          else
32          {
33              // これまで最新だった領域に新しい領域を連結する(enqueue:3-1)
34              top_queue->addr = new_queue;
35              // 新しい領域が最新であるので，最新を指すアドレスに代入する(enqueue:3-2)
36              top_queue = new_queue;
37
38          }
39          r_val = 0;
40      }
41
42      return r_val;
43  }
```

・dequeue 関数

このプログラムでは、リスト構造からキューを取り出す。構造体はグローバル変数で宣言しているため引数はない。

表 13　dequeue 関数

機能	キューからデータを取り出す。
引数	なし
戻り値	int：エラーコード (-200: 正常終了, -201: データが存在しない, -202: 予測しえないエラー)

ソースコード 16　dequeue 関数

```
1   int dequeue()
2   {
3       int r_val = -200;          // 返り値を入れる変数を確保する
4       struct queue *new_bottom; // 新たに最古になる領域を指すポインタを宣言する
5       // 残りキューが 1 個の場合
6       if (top_queue == bottom_queue && bottom_queue != NULL)
7       {
8           // 最古のデータを返り値にするために取り出す(dequeue:1-1)
9           // データを取り出した後のキューを解放する(dequeue:1-2)
10          // 最新のキューがないことを宣言する(dequeue:1-3)
11          // 最古のキューがないことを宣言する(dequeue:1-4)
12          r_val = bottom_queue->val;
13          free(bottom_queue);
14          top_queue = NULL;
15          bottom_queue = NULL;
16
```

```
17      }
18      // 残りのキューが 2 個以上の場合
19      else if (top_queue != NULL && bottom_queue != NULL)
20      {
21          // 最古のデータを返り値にするために取り出す(dequeue:2-1)
22          // 新たに最古になるキューのアドレスを覚えておく(dequeue:2-2)
23          // データを取り出した後のキューを解放する(dequeue:2-3)
24          // 新たに最古のキューになるアドレスを更新する(dequeue:2-4)
25          r_val = bottom_queue->val;
26          new_bottom = bottom_queue->addr;
27          free(bottom_queue);
28          bottom_queue = new_bottom;
29      }
30      // キューが既に空っぽの場合
31      else if (bottom_queue == NULL)
32      {
33          r_val = -201;
34      }
35      // 予測しえないエラー
36      else
37      {
38          r_val = -202;
39      }
40      return r_val;
41  }
```

• showQueue 関数

このプログラムでは、リスト構造の中身を表示する。構造体はグローバル変数で宣言しているため引数はない。

表 14　showQueue 関数

機能	キューの中身を表示する。
引数	なし
戻り値	int：エラーコード (0: 正常終了)

ソースコード 17　showQueue 関数

```
1   int showQueue()
2   {
3       // リスト全体のデータを順に表示する.
4       // データとデータの間に区切り文字「|」を表示する.
5       struct queue *this_queue = bottom_queue;
6       while (this_queue != NULL)
7       {
8           printf("%d|", this_queue->val);
9           this_queue = this_queue->addr;
10      }
```

```
11      }
```

・showResult 関数

このプログラムでは、エラーコードに応じてエラーメッセージを表示する。引数としてエラーコードを受け取り、エラーコードに応じたエラーメッセージを表示する。

表 15　showResult 関数

機能	関数の実行結果を表示する。
引数	int result：エラーコード
戻り値	なし

ソースコード 18　showResult 関数

```
1   void showResult(int result)
2   {
3       // result の値に応じて，対応するエラーメッセージを表示する．
4       switch (result)
5       {
6       case -100:
7           printf("エラー：挿入するデータが 0 以下です．\n");
8           break;
9       case -101:
10          printf("エラー：メモリの確保に失敗しました．\n");
11          break;
12      case -102:
13          printf("エラー：挿入するデータが 0 以下です．\n");
14          break;
15      case -200:
16          printf("エラー：キューが空です．\n");
17          break;
18      case -201:
19          printf("エラー：キューが空です．\n");
20          break;
21      case -202:
22          printf("エラー：予期しないエラーが発生しました．\n");
23          break;
24      default:
25          printf("\n");
26          break;
27      }
28  }
```

・freeQueue 関数

このプログラムでは、bottom_queue のアドレスにあるメモリを開放する関数である。

表 16　freeQueue 関数

機能	キューのメモリを解放する。
引数	なし
戻り値	なし

ソースコード 19　freeQueue 関数

```
1    void freeQueue()
2  {
3      struct queue *this_queue;
4      while (bottom_queue != NULL)
5      {
6          // 今回開放したいキューのアドレスを取得
7          // 解放後に最古になるキューのアドレスを更新
8          // キューの領域を解放する
9          this_queue = bottom_queue;
10         bottom_queue = bottom_queue->addr;
11         free(this_queue);
12     }
13 }
```

・main 関数

キューが正しく動作しているかを確かめるために main 関数を作成する。main 関数はキューの初期化、enqueue、dequeue、showQueue、showResult を行う。リスト構造ではメモリがあふれるまでデータを追加できるため、enqueue のエラーは自然数以外が代入されたとき以外である。

ソースコード 20　main 関数

```
1  int main()
2  {
3      int result;
4      result = enqueue(40);
5      showQueue();
6      printf("<40");
7      showResult(result);
8      result = enqueue(60);
9      showQueue();
10     printf("<60");
11     showResult(result);
12     result = enqueue(10);
13     showQueue();
14     printf("<10");
15     showResult(result);
16     result = enqueue(80);
17     showQueue();
```

```
18      printf("<80");
19      showResult(result);
20      result = enqueue(30);
21      showQueue();
22      printf("<30");
23      showResult(result);
24      result = enqueue(50);
25      showQueue();
26      printf("<50");
27      showResult(result);
28      result = dequeue();
29      showQueue();
30      printf(">%d", result);
31      showResult(result);
32      result = enqueue(1);
33      showQueue();
34      printf("<1");
35      showResult(result);
36      result = enqueue(2);
37      showQueue();
38      printf("<2");
39      showResult(result);
40      result = dequeue();
41      showQueue();
42      printf(">%d", result);
43      showResult(result);
44      result = dequeue();
45      showQueue();
46      printf(">%d", result);
47      showResult(result);
48      result = dequeue();
49      showQueue();
50      printf(">%d", result);
51      showResult(result);
52      result = dequeue();
53      showQueue();
54      printf(">%d", result);
55      showResult(result);
56      result = dequeue();
57      showQueue();
58      printf(">%d", result);
59      showResult(result);
60      result = dequeue();
61      showQueue();
62      printf(">%d", result);
63      showResult(result);
64      result = dequeue();
65      showQueue();
66      printf(">%d", result);
```

```
67      showResult(result);
68      result = dequeue();
69      showQueue();
70      printf(">%d", result);
71      showResult(result);
72      result = enqueue(200);
73      showQueue();
74      printf("<200");
75      showResult(result);
76      result = enqueue(300);
77      showQueue();
78      printf("<300");
79      showResult(result);
80      result = enqueue(-400);
81      showQueue();
82      printf("<-400");
83      showResult(result);
84      result = enqueue(500);
85      showQueue();
86      printf("<500");
87      showResult(result);
88      freeQueue();
89
90      return 0;
91  }
```

5 実行結果

5.1 配列を用いたキューの実装

main 関数を実行した結果を以下に示す。

```
1  1|0|0|0|--> -100:enqueue 成功
2  1|2|0|0|--> -100:enqueue 成功
3  1|2|3|0|--> -100:enqueue 成功
4  1|2|3|4|--> -100:enqueue 成功
5  1|2|3|4|--> -101:エラー:キューが満杯です
6  2|3|4|0|--> 1:dequeue(1)
7  3|4|0|0|--> 2:dequeue(2)
8  4|0|0|0|--> 3:dequeue(3)
9  0|0|0|0|--> 4:dequeue(4)
10 0|0|0|0|--> -201:エラー:キューが空です
```

データがあふれてるときやデータがないときにエラーが出力されていることがわかる。

5.2 リングバッファを用いたキューの実装

main 関数を実行した結果を以下に示す。

```
1  40|<40
```

```
 2  40|60|<60
 3  40|60|10|<10
 4  40|60|10|80|<80
 5  40|60|10|80|30|<30
 6  40|60|10|80|30|50|<50
 7  60|10|80|30|50|>40
 8  60|10|80|30|50|1|<1
 9  60|10|80|30|50|1|2|<2
10  10|80|30|50|1|2|>60
11  80|30|50|1|2|>10
12  30|50|1|2|>80
13  50|1|2|>30
14  1|2|>50
15  2|>1
16  >2
17  >-201エラー:キューが空です.
18  200|<200
19  200|300|<300
20  200|300|<-400エラー:挿入するデータが 0 以下です.
21  200|300|500|<500
```

5.3 リスト構造を用いたキューの実装

main 関数を実行した結果を以下に示す。

```
 1  40|<40
 2  40|60|<60
 3  40|60|10|<10
 4  40|60|10|80|<80
 5  40|60|10|80|30|<30
 6  40|60|10|80|30|50|<50
 7  60|10|80|30|50|>40
 8  60|10|80|30|50|1|<1
 9  60|10|80|30|50|1|2|<2
10  10|80|30|50|1|2|>60
11  80|30|50|1|2|>10
12  30|50|1|2|>80
13  50|1|2|>30
14  1|2|>50
15  2|>1
16  >2
17  >-201エラー:キューが空です.
18  200|<200
19  200|300|<300
20  200|300|<-400エラー:挿入するデータが 0 以下です.
21  200|300|500|<500
```

6 考察

6.1 配列を用いたキューの実装

配列を用いたキューの実装では、配列のサイズを決めているため、あふれるとエラーが出力される。また、データがないときにもエラーが出力された。このことが実験結果からわかるため、正しくキューが実装されて

いることがわかる。

6.2　リングバッファを用いたキューの実装

リングバッファを用いたキューの実装では、配列のサイズを決めているため、あふれるとエラーが出力される。また、データがないときにもエラーが出力された。このことが実験結果からわかるため、正しくキューが実装されていることがわかる。

6.3　リスト構造を用いたキューの実装

キューが空の場合、enqueue するデータが 0 以下の倍のエラーが出力されているのがわかる。このことからリスト構造を用いたキューの実装が成功していると思われる。また、メイン関数のプログラムを変更してメモリを確保できない場合のエラーを出力するようにした。出力結果は以下の通りである。

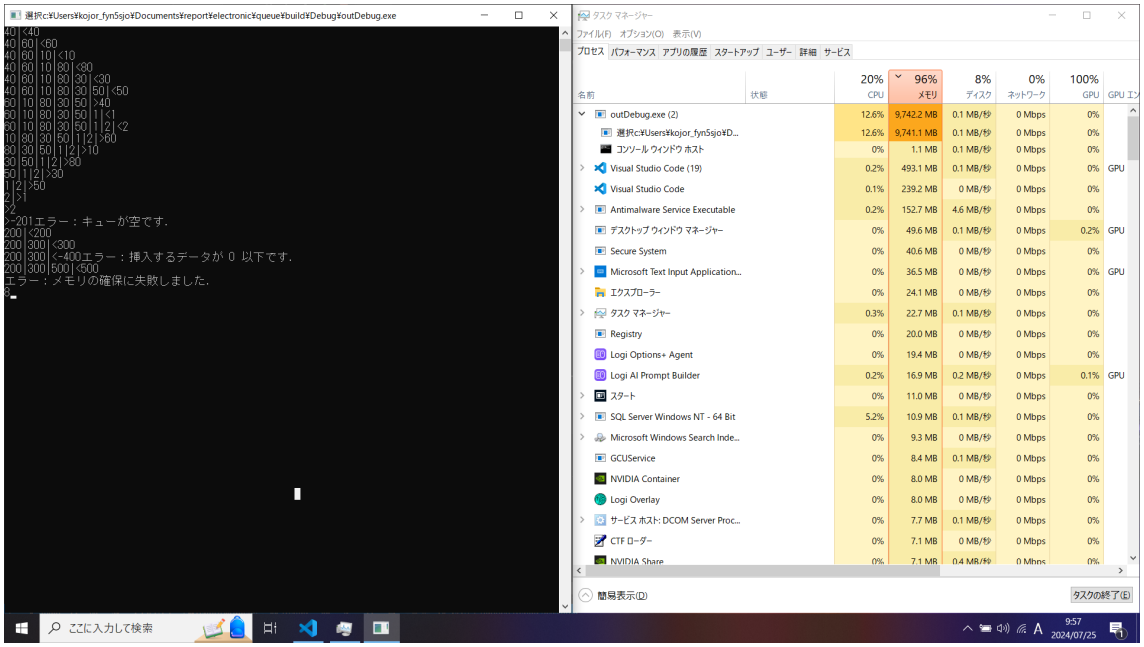

図 1　メモリ確保エラー

メモリの確保が失敗した後にパソコンの画面が一瞬真っ黒になったため、そこで実験を終了した。

7　付録: 今回使用したプログラム

ソースコード 21　hairetu.c

```
1  #include <stdio.h>
2  #define QUEUE_SIZE 4
3  int queue[QUEUE_SIZE];
4  int quantity = 0;
5
```

```
6  // #define MAX_SIZE 1000000 // キューの最大サイズ
7  int enqueue(int data);       // キューにデータを追加する関数
8  int dequeue();               // キューからデータを取り出す関数
9  int initQueue();             // キューを初期化する関数
10 int showQueue();             // キューの中身を表示する関数
11 void showResult(int result); // 結果を表示する関数
12
13 // メイン関数
14 int main()
15 {
16     // キューを初期化する
17     initQueue();
18     // キューにデータを追加する
19     showResult(enqueue(1));
20     showResult(enqueue(2));
21     showResult(enqueue(3));
22     showResult(enqueue(4));
23     showResult(enqueue(5));
24     // キューの中身を表示する
25     // showQueue();
26     // キューからデータを取り出す
27     showResult(dequeue());
28     showResult(dequeue());
29     showResult(dequeue());
30     showResult(dequeue());
31     showResult(dequeue());
32     // キューの中身を表示する
33     // showQueue();
34     return 0;
35 }
36
37 int enqueue(int data)
38 {
39     // 残り領域があるか確認する
40     if (quantity >= QUEUE_SIZE)
41     {
42         return -101;
43     }
44     // データが自然数か確認する
45     if (data < 0)
46     {
47         return -102;
48     }
49     // 配列のキューにデータを保存する
50     queue[quantity] = data;
51     // キューのポインタをインクリメントする
52     quantity++;
53     // 返り値を返す
54     return -100;
```

```
55  }
56
57  int dequeue(void)
58  {
59      // データが存在するかどうか確認する.
60      if (quantity <= 0)
61      {
62          return -201;
63      }
64      // キューからデータをとりだす.
65      int data = queue[0];
66      // データの個数カウントを減らす
67      quantity--;
68      // 取り出されたデータ部分を埋めるように再構築する
69      for (int i = 0; i < quantity; i++)
70      {
71          queue[i] = queue[i + 1];
72      }
73      // 空き領域を初期化する.
74      queue[quantity] = 0;
75      // データもしくは返り値を返す
76      return data;
77  }
78
79  int initQueue()
80  {
81      // キューのデータを入れる配列をすべて 0 に初期化する.
82      for (int i = 0; i < QUEUE_SIZE; i++)
83      {
84          queue[i] = 0;
85      }
86      // 格納データ個数を 0 に初期化する.
87      quantity = 0;
88      return 0;
89  }
90
91  int showQueue()
92  {
93      // 配列全体のデータを順に表示する.
94      // データとデータの間に区切り文字「|」を表示する.
95      for (int i = 0; i < QUEUE_SIZE; i++)
96      {
97          printf("%d|", queue[i]);
98      }
99      return 0;
100 }
101
102 void showResult(int result)
103 {
```

```
104     // result の値に応じて，対応するエラーメッセージを表示する．
105     showQueue();
106     printf("--> %d:", result);
107     switch (result)
108     {
109     case -100:
110         printf("enqueue 成功\n");
111         break;
112     case -101:
113         printf("エラー：キューが満杯です\n");
114         break;
115     case -102:
116         printf("エラー：データが自然数ではありません\n");
117         break;
118     case -201:
119         printf("エラー：キューが空です\n");
120         break;
121     default:
122         if (result < 0)
123         {
124             printf("エラー：不明なエラーです\n");
125         }
126         else
127         {
128             printf("dequeue(%d)\n", result);
129         }
130         break;
131     }
132 }
```

ソースコード 22　ring.c

```
1  #define QUEUE_SIZE 5
2  struct queue
3  {
4      int array[QUEUE_SIZE]; // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf
       ^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdz^^ef^^bf^^bd^^ef^^bf^^bd
5      int wp;                // ^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^c9^^83f^^ef^^bf^^bd[^^
       ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd
       ^^ea^^8f^^8a^^ef^^bf^^bd^^cc^^94z^^ef^^bf^^bd^^ef^^bf^^bd^^d4^^8d^^ef^^bf^^bd
6      int quantity;          // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^cc^^8c
       ^^c2^^90^^ef^^bf^^bd
7  };
8
9  int enqueue(struct queue *obj, int data)
10 {
11     // ^^ef^^bf^^bdc^^ef^^bf^^bd^^ef^^bf^^bd^^cc^^88悪^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd
       ^^e9^^82^^a9^^ef^^bf^^bdm^^ef^^bf^^bdF^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^
       bdD
```

```
12      // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf
          ^^bd^^ef^^bf^^bdR^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdm^^ef^^
          bf^^bdF^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
13      // ^^ef^^bf^^bdz^^ef^^bf^^bd^^ef^^bf^^bd^^c9^^83f^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^
          bd^^ef^^bf^^bd^^db^^91^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
14      // ^^ef^^bf^^bd^^c5^^90V^^ef^^bf^^bd^^cc^^83f^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^
          ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^
          ef^^bf^^bd^^ea^^8f^^8a^^ef^^bf^^bd^^cc^^8e^^ef^^bf^^bd^^ef^^bf^^bd^^cc^^8f^^ea^^8f
          ^^8a^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdw^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^e6
          ^^82^^a4^^ef^^bf^^bd^^c9^^92l^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdX^^ef^^bf^^bdV^^ef
          ^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
15      // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^cc^^8c^^c2^^90^^ef^^bf^^bd^^ef
          ^^bf^^bdJ^^ef^^bf^^bdE^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdg^^ef^^bf^^bd^^f0^^91^^9d
          ^^82^^e2^^82^^b7^^ef^^bf^^bdD
16      // ^^ef^^bf^^bd^^d4^^82^^ef^^bf^^bdl^^ef^^bf^^bd^^ef^^bf^^bd^^d4^^82^^ef^^bf^^bd^^ef^^
          bf^^bdD
17      if (obj->quantity >= QUEUE_SIZE)
18      {
19          return -101;
20      }
21      if (data < 0)
22      {
23          return -102;
24      }
25      obj->array[obj->wp] = data;
26      obj->wp = (obj->wp + 1) % QUEUE_SIZE;
27      obj->quantity++;
28      return -100;
29  }
30  int dequeue(struct queue *obj)
31  {
32      // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf
          ^^bd^^ef^^bf^^bd^^dd^^82^^ef^^bf^^bd^^ef^^bf^^bd^^e9^^82^^a9^^ef^^bf^^bd^^c7^^82^^ef
          ^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdm^^ef^^bf^^bdF^^ef^^bf^^bd^^ef^^bf^^bd^^
          ef^^bf^^bd^^ef^^bf^^bdD
33      // ^^ef^^bf^^bdL^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf
          ^^bd^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^c6
          ^^82^^e8^^82^^be^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
34      // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^cc^^8c^^c2^^90^^ef^^bf^^bd^^ef
          ^^bf^^bd^^cc^^83J^^ef^^bf^^bdE^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdg^^ef^^bf^^bd^^ef
          ^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^e7^^82^^b7^^ef^^bf^^bdD
35      // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf
          ^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^cd^^95^^d4^^82^^ef^^bf^^bdl^^ef^^bf^^bd^^
          ef^^bf^^bd^^d4^^82^^ef^^bf^^bd^^ef^^bf^^bdD
36      if (obj->quantity <= 0)
37      {
38          return -201;
39      }
40      int data = obj->array[0];
```

```
41      obj->quantity--;
42      obj->array[(obj->wp+QUEUE_SIZE - obj->quantity - 1) % QUEUE_SIZE] = 0;
43      return data;
44  }
45  int initqueue(struct queue *obj)
46  {
47      // ^^ef^^bf^^bdL^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^cc^^83f^^ef^^bf^^bd
          [^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^
          bdz^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^d7
          ^^82^^ef^^bf^^bd 0 ^^ef^^bf^^bd^^c9^^8f^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf
          ^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
48      // ^^ef^^bf^^bdL^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^cc^^83f^^ef^^bf^^bd
          [^^ef^^bf^^bd^^^ef^^bf^^bdi^^ef^^bf^^bd[^^ef^^bf^^bd^^c2^^90^^ef^^bf^^bd^^ef^^bf^^bd
          ^^ef^^bf^^bd 0 ^^ef^^bf^^bd^^c9^^8f^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd
          ^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd
49      // ^^ef^^bf^^bdL^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^ef^^bf^^bd wp ^^ef
          ^^bf^^bd|^^ef^^bf^^bdC^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd
          0 ^^ef^^bf^^bd^^c9^^8f^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^
          ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
50      for (int i = 0; i < QUEUE_SIZE; i++)
51      {
52          obj->array[i] = 0;
53      }
54      obj->quantity = 0;
55      obj->wp = 0;
56      return 0;
57  }
58  int showQueue(struct queue *obj)
59  {
60      // ^^ef^^bf^^bdz^^ef^^bf^^bd^^ef^^bf^^bdS^^ef^^bf^^bd^^cc^^82^^cc^^83f^^ef^^bf^^bd[^^
          ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^c9^^95\^^ef
          ^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
61      // ^^ef^^bf^^bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^c6^^83f^^ef^^bf^^bd[^^ef^^bf^^
          bd^^^ef^^bf^^bd^^cc^^8a^^d4^^82^^c9^^8b^^ef^^bf^^bd^^d8^^82^^e8^^95^^b6^^ef^^bf^^bd
          ^^ef^^bf^^bd^^ef^^bf^^bdu|^^ef^^bf^^bdv^^ef^^bf^^bd^^ef^^bf^^bd\^^ef^^bf^^bd^^ef^^bf
          ^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
62      for (int i = 0; i < QUEUE_SIZE; i++)
63      {
64          printf("%d", obj->array[i]);
65          if (i < QUEUE_SIZE - 1)
66          {
67              printf("|");
68          }
69      }
70      return 0;
71  }
72  void showResult(int result)
73  {
74      // result ^^ef^^bf^^bd^^cc^^92l^^ef^^bf^^bd^^c9^^89^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^
```

```
        bd^^ef^^bf^^bd^^c4^^81C^^ef^^bf^^bd^^ce^^89^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef
        ^^bf^^bd^^ef^^bf^^bdG^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^ef^^bf^^bd^^
        ef^^bf^^bdb^^ef^^bf^^bdZ^^ef^^bf^^bd[^^ef^^bf^^bdW^^ef^^bf^^bd^^ef^^bf^^bd\^^ef^^bf
        ^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdD
75      switch (result)
76      {
77      case -100:
78          printf("^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bdI^^ef^^bf^^bd^^ef^^bf^^bd\
              n");
79          break;
80      case -101:
81
82          printf("^^ef^^bf^^bdG^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bdF^^ef^^bf^^
              bdL^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef
              ^^bf^^bd^^ef^^bf^^bdt^^ef^^bf^^bd^^c5^^82^^ef^^bf^^bd\n");
83          break;
84      case -102:
85          printf("^^ef^^bf^^bdG^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bdF^^ef^^bf^^
              bdf^^ef^^bf^^bd[^^ef^^bf^^bd^^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^
              ef^^bf^^bdR^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^c5^^82^^cd^^82^^ef^^bf^^bd^^ef
              ^^bf^^bd^^ef^^bf^^bd^^dc^^82^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd\n");
86          break;
87      case -201:
88          printf("^^ef^^bf^^bdG^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bdF^^ef^^bf^^
              bdL^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd[^^ef^^bf^^bd^^ef^^bf^^bd^^ef^^bf^^bd^^ef
              ^^bf^^bd^^c5^^82^^ef^^bf^^bd\n");
89          break;
90      default:
91          printf("\n");
92          break;
93      }
94  }
95
96  int main(void)
97  {
98
99      struct queue obj;
100     int result;
101     initqueue(&obj);
102     result = enqueue(&obj,40);
103     showQueue(&obj);
104     printf("<40");
105     showResult(result);
106     result = enqueue(&obj,60);
107     showQueue(&obj);
108     printf("<60");
109     showResult(result);
110     result = enqueue(&obj,10);
111     showQueue(&obj);
```

```
112     printf("<10");
113     showResult(result);
114     result = enqueue(&obj,80);
115     showQueue(&obj);
116     printf("<80");
117     showResult(result);
118     result = enqueue(&obj,30);
119     showQueue(&obj);
120     printf("<30");
121     showResult(result);
122     result = enqueue(&obj,50);
123     showQueue(&obj);
124     printf("<50");
125     showResult(result);
126     result = dequeue(&obj);
127     showQueue(&obj);
128     printf(">%d", result);
129     showResult(result);
130     result = enqueue(&obj,1);
131     showQueue(&obj);
132     printf("<1");
133     showResult(result);
134     result = enqueue(&obj,2);
135     showQueue(&obj);
136     printf("<2");
137     showResult(result);
138     result = dequeue(&obj);
139     showQueue(&obj);
140     printf(">%d", result);
141     showResult(result);
142     result = dequeue(&obj);
143     showQueue(&obj);
144     printf(">%d", result);
145     showResult(result);
146     result = dequeue(&obj);
147     showQueue(&obj);
148     printf(">%d", result);
149     showResult(result);
150     result = dequeue(&obj);
151     showQueue(&obj);
152     printf(">%d", result);
153     showResult(result);
154     result = dequeue(&obj);
155     showQueue(&obj);
156     printf(">%d", result);
157     showResult(result);
158     result = dequeue(&obj);
159     showQueue(&obj);
160     printf(">%d", result);
```

```
161     showResult(result);
162     result = dequeue(&obj);
163     showQueue(&obj);
164     printf(">%d", result);
165     showResult(result);
166     result = dequeue(&obj);
167     showQueue(&obj);
168     printf(">%d", result);
169     showResult(result);
170     result = enqueue(&obj,200);
171     showQueue(&obj);
172     printf("<200");
173     showResult(result);
174     result = enqueue(&obj,300);
175     showQueue(&obj);
176     printf("<300");
177     showResult(result);
178     result = enqueue(&obj,-400);
179     showQueue(&obj);
180     printf("<-400");
181     showResult(result);
182     result = enqueue(&obj,500);
183     showQueue(&obj);
184     printf("<500");
185     showResult(result);
186     return 0;
187 }
```

ソースコード 23　list.c

```
1  #include <malloc.h>
2  #include <stdio.h>
3  // データと次のキューのアドレスを指す構造体の宣言
4
5  struct queue
6  {
7      int val;
8      struct queue *addr;
9  };
10 struct queue *bottom_queue = NULL; // 最古のキューのアドレスを記憶しておくポインタ
11 struct queue *top_queue = NULL;    // 最新のキューのアドレスを記憶しておくポインタ
12 int enqueue(int data)
13 {
14     int r_val = -100;                          // 返り値を宣言する
15     struct queue *new_queue;                   // 新しく確保する領域（構造体）へのポイン
       タを宣言する
16     if (data < 0 /*挿入するデータが 0 以下であったとき*/) // 0 以下のデータが入力された時
17     {
18         r_val = -102; // 返り値を設定する
```

```
19      }
20      // 追加する領域を確保する
21      /*malloc で構造体のオブジェクトの領域を確保して，確保失敗した時*/
22      else if ((new_queue = (struct queue *)malloc(sizeof(struct queue))) == NULL)
23      {
24          // 確保できなかったとき
25          r_val = -101;
26      }
27      else
28      {
29          // 新しいキューのデータ値として引数のデータを入れる(enqueue:2-1)
30          new_queue->val = data;
31          // 新しいキューのアドレス値として NULL を入れる(enqueue:2-2)
32          new_queue->addr = NULL;
33          // まだ 1 個のキューもない時
34          if (bottom_queue == NULL)
35          {
36              // 新しいキューが最古であるので，最古のキューを指すアドレスを更新する(enqueue
                    :4-1)
37              bottom_queue = new_queue;
38              // 新しいキューが最新であるので，最新のキューを指すアドレスを更新する(enqueue
                    :4-2)
39              top_queue = new_queue;
40          }
41          // 1 個以上のキューがあるとき
42          else
43          {
44              // これまで最新だった領域に新しい領域を連結する(enqueue:3-1)
45              top_queue->addr = new_queue;
46              // 新しい領域が最新であるので，最新を指すアドレスに代入する(enqueue:3-2)
47              top_queue = new_queue;
48
49          }
50          r_val = 0;
51      }
52
53      return r_val;
54  }
55  int dequeue()
56  {
57      int r_val = -200;          // 返り値を入れる変数を確保する
58      struct queue *new_bottom; // 新たに最古になる領域を指すポインタを宣言する
59      // 残りキューが 1 個の場合
60      if (top_queue == bottom_queue && bottom_queue != NULL)
61      {
62          // 最古のデータを返り値にするために取り出す(dequeue:1-1)
63          // データを取り出した後のキューを解放する(dequeue:1-2)
64          // 最新のキューがないことを宣言する(dequeue:1-3)
65          // 最古のキューがないことを宣言する(dequeue:1-4)
```

```
66          r_val = bottom_queue->val;
67          free(bottom_queue);
68          top_queue = NULL;
69          bottom_queue = NULL;
70
71      }
72      // 残りのキューが 2 個以上の場合
73      else if (top_queue != NULL && bottom_queue != NULL)
74      {
75          // 最古のデータを返り値にするために取り出す(dequeue:2-1)
76          // 新たに最古になるキューのアドレスを覚えておく(dequeue:2-2)
77          // データを取り出した後のキューを解放する(dequeue:2-3)
78          // 新たに最古のキューになるアドレスを更新する(dequeue:2-4)
79          r_val = bottom_queue->val;
80          new_bottom = bottom_queue->addr;
81          free(bottom_queue);
82          bottom_queue = new_bottom;
83      }
84      // キューが既に空っぽの場合
85      else if (bottom_queue == NULL)
86      {
87          r_val = -201;
88      }
89      // 予測しえないエラー
90      else
91      {
92          r_val = -202;
93      }
94      return r_val;
95  }
96  int showQueue()
97  {
98      // リスト全体のデータを順に表示する．
99      // データとデータの間に区切り文字「|」を表示する．
100     struct queue *this_queue = bottom_queue;
101     while (this_queue != NULL)
102     {
103         printf("%d|", this_queue->val);
104         this_queue = this_queue->addr;
105     }
106 }
107 void showResult(int result)
108 {
109     // result の値に応じて，対応するエラーメッセージを表示する．
110     switch (result)
111     {
112     case -100:
113         printf("エラー：挿入するデータが 0 以下です．\n");
114         break;
```

```
115     case -101:
116         printf("エラー：メモリの確保に失敗しました．\n");
117         break;
118     case -102:
119         printf("エラー：挿入するデータが 0 以下です．\n");
120         break;
121     case -200:
122         printf("エラー：キューが空です．\n");
123         break;
124     case -201:
125         printf("エラー：キューが空です．\n");
126         break;
127     case -202:
128         printf("エラー：予期しないエラーが発生しました．\\n");
129         break;
130     default:
131         printf("\n");
132         break;
133     }
134 }
135 void freeQueue()
136 {
137     struct queue *this_queue;
138     while (bottom_queue != NULL)
139     {
140         // 今回開放したいキューのアドレスを取得
141         // 解放後に最古になるキューのアドレスを更新
142         // キューの領域を解放する
143         this_queue = bottom_queue;
144         bottom_queue = bottom_queue->addr;
145         free(this_queue);
146     }
147 }
148 int main()
149 {
150     int result;
151     result = enqueue(40);
152     showQueue();
153     printf("<40");
154     showResult(result);
155     result = enqueue(60);
156     showQueue();
157     printf("<60");
158     showResult(result);
159     result = enqueue(10);
160     showQueue();
161     printf("<10");
162     showResult(result);
163     result = enqueue(80);
```

```
164     showQueue();
165     printf("<80");
166     showResult(result);
167     result = enqueue(30);
168     showQueue();
169     printf("<30");
170     showResult(result);
171     result = enqueue(50);
172     showQueue();
173     printf("<50");
174     showResult(result);
175     result = dequeue();
176     showQueue();
177     printf(">%d", result);
178     showResult(result);
179     result = enqueue(1);
180     showQueue();
181     printf("<1");
182     showResult(result);
183     result = enqueue(2);
184     showQueue();
185     printf("<2");
186     showResult(result);
187     result = dequeue();
188     showQueue();
189     printf(">%d", result);
190     showResult(result);
191     result = dequeue();
192     showQueue();
193     printf(">%d", result);
194     showResult(result);
195     result = dequeue();
196     showQueue();
197     printf(">%d", result);
198     showResult(result);
199     result = dequeue();
200     showQueue();
201     printf(">%d", result);
202     showResult(result);
203     result = dequeue();
204     showQueue();
205     printf(">%d", result);
206     showResult(result);
207     result = dequeue();
208     showQueue();
209     printf(">%d", result);
210     showResult(result);
211     result = dequeue();
212     showQueue();
```

```
213     printf(">%d", result);
214     showResult(result);
215     result = dequeue();
216     showQueue();
217     printf(">%d", result);
218     showResult(result);
219     result = enqueue(200);
220     showQueue();
221     printf("<200");
222     showResult(result);
223     result = enqueue(300);
224     showQueue();
225     printf("<300");
226     showResult(result);
227     result = enqueue(-400);
228     showQueue();
229     printf("<-400");
230     showResult(result);
231     result = enqueue(500);
232     showQueue();
233     printf("<500");
234     showResult(result);
235 
236     for(long i = 0; i < 10000000000;i++){
237         result = enqueue(i);
238         // showQueue();
239         // printf("<%ld",i);
240         if(result != 0){
241             showResult(result);
242             break;
243         }
244     }
245 
246     freeQueue();
247 
248     return 0;
249 }
```