上級プログラミング第一回レポート課題

2019/05/03

us162039　梶田悠

2019

目次

[レポートの課題内容 2](#_Toc7807334)

[取り組んだ内容 2](#_Toc7807335)

[コード 2](#_Toc7807336)

[基礎課題D 6](#_Toc7807337)

[問題文 6](#_Toc7807338)

[改良例 1 6](#_Toc7807339)

[改良例 2 6](#_Toc7807340)

[プログラムの改良 7](#_Toc7807341)

[前述の改良例１ 7](#_Toc7807342)

[デバッグの効率化 10](#_Toc7807343)

[食品の追加の記述を関数に切り分け 13](#_Toc7807344)

[前述の改良例２ 16](#_Toc7807345)

# レポートの課題内容

第3 回演習で完成させたプログラムをもとに以下の課題を順番に進める。基礎課題 ABCは全て完成させること。発展課題Dについては余裕があれば挑戦すること。

## 取り組んだ内容

課題A、B、C、Dに取り組みました。

コードは[github](https://github.com/HarukaKajita/jpro_report01)にアップロードしてあるので、レポートのコードが読みづらい場合は適宜参照して下さい。

## コード

基礎課題C完了時点でのコードを以下に掲載する。このコードを課題Dにおいて改良していく。

コード 3 reort01\_C.cpp

|  |
| --- |
| #include<iostream>  #include<iomanip>  #include<fstream>  #include<cstdlib>  using namespace std;  class Food {  private:  string name; //食品名  int number; //個数  public:  Food(); //コンストラクタ  Food(string, int); //nameとnumberを設定可能なコンストラクタ  void printFood(); //nameとnumberを出力する  void increment(); //numberを1増やす  void decrement(); //numberを1減らす  bool check\_name(string);//食品名が一緒かどうかチェックする  string get\_name() { return name; }  int get\_number() { return number; }  };  Food::Food() {//コンストラクタ(食品名は空文字列""に、個数は0に初期化)  name = "";  number = 0;  }  Food::Food(string str, int u) {//コンストラクタ(食品名はstrに、個数はuに初期化)  name = str;  number = u;  }  void Food::printFood() {  cout << name << ":" << number << endl;  }  void Food::increment() {  number++;  }  void Food::decrement() {  number--;  }  bool Food::check\_name(string str) {  if (name == str) {  return true;  }  else {  return false;  }  }  class Storage {  private:  Food foods[300];  int cnt;  public:  Storage() { cnt = 0; }  void input\_foods();  void output\_foods();  void printStorage();  void sort\_by\_number();  void sort\_by\_name();  };  void Storage::input\_foods() {  //ファイルからの入力の準備  cout << "Input Foods." << endl;;  cout << "Filename? :";  string fname;  cin >> fname;  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  int i;  for (i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].increment();  break;  }  }  if (i == cnt) {  foods[i] = Food(current\_name, 1);  cnt++;  }  }  }  void Storage::output\_foods() {  //ファイルからの入力の準備  cout << "Output Foods." << endl;;  cout << "Filename? :";  string fname;  cin >> fname;  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  //一行ずつ読んで配列から該当食品を探しデクリメント  for (int i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].decrement();  break;  }  }  }  }  void Storage::printStorage() {  //食品リストの表示  cout << endl;  cout << "////////////// All Food List ///////////////" << endl;  for (int i = 0; i < cnt; i++) {  cout << "[" << setw(2) << i + 1 << "] ";  foods[i].printFood();  }  cout << "////////////// All Food List END ///////////////" << endl;  cout << endl;  }  void Storage::sort\_by\_number()  {  for (int i = 0; i < cnt - 1; i++) {  for (int j = i + 1; j < cnt; j++) {  //食品の数が昇順になっていなければswapFlagをtrueにする  bool swapFlag =  　　foods[j].get\_number() < foods[i].get\_number();  if (!swapFlag) continue;  //入れ替え  swap(foods[i], foods[j]);  }  }  }  void Storage::sort\_by\_name()  {  for (int i = 0; i < cnt - 1; i++) {  for (int j = i + 1; j < cnt; j++) {  //アルファベット順になっていなければswapFlagをtrueにする  bool swapFlag = foods[j].get\_name() < foods[i].get\_name();  if (!swapFlag) continue;  //入れ替え  swap(foods[i], foods[j]);  }  }  }  int main() {  Storage s1;  s1.input\_foods();  s1.output\_foods();  s1.sort\_by\_name();  s1.printStorage();  } |

# 基礎課題D

## 問題文

基礎課題Cまでで完成させたプログラムを更に改良することを考える。Storageクラスに追加で必要とされそうな機能を考え、プログラムを追加・修正すること。以下に二つ追加・修正の例を載せるが、これら以外に自分なりの改良を考えてもよい。レポートには、プログラムと実行結果だけではなく、何の目的で、どのような追加・修正をしたかを具体的に記述すること。

### 改良例 1

食品を取り出した結果、個数が 0 となるものが存在する。メンバ関数 reduction\_foods()により、0 個の食品をリストから削除する機能を実装すること。結果のリストは食品名でソート後に表示すること。レポートには、どのように考えて reduction\_foods()を実装したかアイデアも記述すること。

### 改良例 2

foods\_output.txt の内容によっては、食品を取り出そうとしても存在しなかったり、存在しても個数がマイナスとなってしまう場合 がある。そのような状況が生じたときに適切に処理するためには、プログラムをどのように変更すべきかを考えて、実装と実験をしてみること。レポートには、どのような処理方法を考えて実装したか具体的に説明すること。

## プログラムの改良

### 前述の改良例１

##### 考察

　decrement()を呼び出した際に個数が0になり得るので、食品リストから食品を削除するreduction\_foods()関数を呼び出すのはdecrement()関数内、もしくはdecrement()を呼び出しているoutput\_foods()関数内になると考えた。Decrement()関数内で呼び出す場合はFoodクラスがStorageクラスのインスタンスの参照を持つ必要がある。食べ物というモノが倉庫というモノの参照を持つという状況はあまり直感的でないという点、Foodクラスが新たにメンバ変数を持つことでFoodクラスのインスタンスの為に確保されるメモリサイズが微量だが増えるという点からouput\_foods()関数内からreduction\_foods()関数を呼び出す実装を選択した。

　具体的には、decrement()関数の呼び出し後にnumber変数の値が0であればreduction\_foods()関数を呼び出すという流れで実装している。reduction\_foods()関数の具体的な実装は、foods配列から削除したい要素のインデックスを引き数とし、その要素以降の要素を一つ後の要素と入れ替える事で削除したい要素を配列の有効な部分（インデックスが0からcnt-1までの範囲）の末尾に移動するというものである。

ouput\_foods()関数内ではreduction\_foods()関数呼び出し後に食品数を表すcnt変数をデクリメントしている点も注意するべきである。上記の実装だと要素が末尾に移動しただけで、printStorage()関数などでfoods配列の要素を処理すると削除したい要素の処理の対象になってしまうのでcnt変数をデクリメントする事で削除したい要素がfoods配列の有効な部分に含まれないようにしている。

##### コード

コード 4　report\_01D01.cpp

|  |
| --- |
| class Storage {  private:  Food foods[300];  int cnt;  public:  Storage() { cnt = 0; }  void input\_foods();  void output\_foods();  void printStorage();  void sort\_by\_number();  void sort\_by\_name();  void reduction\_foods(int);  };  void Storage::input\_foods() {  //ファイルからの入力の準備  cout << "Input Foods." << endl;;  cout << "Filename? :";  string fname;  cin >> fname;    ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  int i;  for (i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].increment();  break;  }  }  if (i == cnt) {  foods[i] = Food(current\_name, 1);  cnt++;  }  }  }  void Storage::output\_foods() {  //ファイルからの入力の準備  cout << "Output Foods." << endl;;  cout << "Filename? :";  string fname;  cin >> fname;  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  //一行ずつ読んで配列から該当食品を探しデクリメント  for (int i = 0; i < cnt; i++) {  if (!foods[i].check\_name(current\_name)) continue;  foods[i].decrement();  //食品の在庫が0になったらリストから削除  if (foods[i].get\_number() == 0) {  reduction\_foods(i);  cnt--;  }  break;  }  }  }  //foods配列から引数で指定した要素番号の要素を削除する  void Storage::reduction\_foods(int index) {  //削除する要素以降の要素を一つ後の要素で上書きして要素を詰める  for (int i = index; i < cnt-1; i++) {  //次の要素ので上書き  swap(foods[i], foods[i + 1]);  }  } |

### デバッグの効率化

##### 考察

　現状のコードだとプログラムの挙動を確認する為に実行すると、入力するtxtファイルのファイル名をキーボードから入力する必要があり手間がかかる。挙動確認の効率化の為にディレクティブを使用し自動でファイル名を入力するコードと、現状の通りファイル名を手動で入力するコードを簡単に切り替え出来るように改良できると良いと考えた。

　コードの先頭付近に#define \_DEBUGを記述し、Storage::input\_foods()とStorage::output\_foods()のファイル名入力の記述付近に#ifdefによる分岐を記述した。これによって#define \_DEBUGの行をコメントアウト・アンコメントアウトするだけで入力方式の切り替えを実現した。

##### コード

コード 5 report01\_D02

|  |
| --- |
| #include<iostream>  #include<iomanip>  #include<fstream>  #include<cstdlib>  //デバッグ時のディレクティブ定義  #define \_DEBUG  using namespace std;  //中略  void Storage::input\_foods() {  //ファイルからの入力の準備  cout << "Input Foods." << endl;;  cout << "Filename? :";  string fname;  #ifdef \_DEBUG  fname = "foods\_input.txt";  #else  cin >> fname;  #endif // \_DEBUG  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  int i;  for (i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].increment();  break;  }  }  if (i == cnt) {  foods[i] = Food(current\_name, 1);  cnt++;  }  }  }  void Storage::output\_foods() {  //ファイルからの入力の準備  cout << "Output Foods." << endl;;  cout << "Filename? :";  string fname;  #ifdef \_DEBUG  fname = "foods\_output.txt";  #else  cin >> fname;  #endif // \_DEBUG  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  //一行ずつ読んで配列から該当食品を探しデクリメント  for (int i = 0; i < cnt; i++) {  if (!foods[i].check\_name(current\_name)) continue;  foods[i].decrement();  //食品の在庫が0になったらリストから削除  if (foods[i].get\_number() == 0) {  reduction\_foods(i);  cnt--;  }  break;  }  }  } |

### 食品の追加の記述を関数に切り分け

##### 考察

　前述の改良例１の改良でoutput\_foods()関数内でreduction\_foods()関数を呼び出した後にcnt変数をデクリメントするコードoutput\_foods()関数内に記述した。デクリメントのコードをoutput\_foods()関数内に記述したのは、input\_foods()関数内にcnt変数をインクリメントするコードが既に記述されていた為、記述する位置を統一させる意図があった。しかし、reduction\_foods()関数を呼び出す時はすなわち食品数が減る時なので必ずデクリメント処理が必要になると考えられる。なので、デクリメントのコードはreduction\_foods()関数内に記述した方がデクリメントのコードを書き忘れるリスクがなくなると考え、実際にそうするように修正した。それに伴って統一性を意識してinput\_foods()関数内に記述されていた食品の追加のコードをreduction\_foods()関数のように関数化し処理を切り離した。

##### コード

コード 6 report01\_D03

|  |
| --- |
| class Storage {  private:  Food foods[300];  int cnt;  public:  Storage() { cnt = 0; }  void input\_foods();  void output\_foods();  void printStorage();  void sort\_by\_number();  void sort\_by\_name();  void increase\_foods(int, string, int);  void reduction\_foods(int);  };  void Storage::input\_foods() {  //ファイルからの入力の準備  cout << "Input Foods." << endl;;  cout << "Filename? :";  string fname;  #ifdef \_DEBUG  fname = "foods\_input.txt";  #else  cin >> fname;  #endif // \_DEBUG  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  int i;  for (i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].increment();  break;  }  }  if (i == cnt) {  increase\_foods(i, current\_name, 1);  }  }  }  void Storage::output\_foods() {  //ファイルからの入力の準備  cout << "Output Foods." << endl;;  cout << "Filename? :";  string fname;  #ifdef \_DEBUG  fname = "foods\_output.txt";  #else  cin >> fname;  #endif // \_DEBUG  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  //一行ずつ読んで配列から該当食品を探しデクリメント  for (int i = 0; i < cnt; i++) {  if (!foods[i].check\_name(current\_name)) continue;  foods[i].decrement();  //食品の在庫が0になったらリストから削除  if (foods[i].get\_number() == 0) {  reduction\_foods(i);    }  break;  }  }  }  void Storage::increase\_foods(int index, string foodName, int num) {  foods[index] = Food(foodName, num);  cnt++;  }  //foods配列から引数で指定した要素番号の要素を削除する  void Storage::reduction\_foods(int index) {  //削除する要素以降の要素を一つ後の要素で上書きして要素を詰める  for (int i = index; i < cnt - 1; i++) {  //次の要素ので上書き  swap(foods[i], foods[i + 1]);  }  cnt--;  } |

### 前述の改良例２

##### 考察

　foods\_output.txtの内容次第でFoodクラスのインスタンスのnumber変数が負の値になり得る問題については、改良例１の改良によってnumber変数が０になった時点でfoods配列の有効な部分から除外され削除したように扱われる為心配する必要はない。

　次に取り出そうとした食品が存在しなかった場合について考える。ユーザーが意図した状況になっていない可能性が高いので、ユーザーに対してそれを周知する事が適切であると考える。その直後にユーザーは食品を必要な分足してから、足りていなかった分取り出そうとする状況が予想できるので、食品を取り出す処理の直後に、どの食品がどれくらい足りていなかったかをまとめて表示するという仕様が適切と考え実装した。

　具体的にはreduction\_foods()関数内に足りていない食品についての情報を格納するFoodクラスの配列を用意し、各食品の取り出し処理の際に食品が存在しなかった場合にこの配列に情報を更新し、全食品の取り出し処理が完了した段階でこの配列の情報を標準出力するプログラムを実装した。

##### コード

以下が最終版のコードである。

コード 7 report01\_D04

|  |
| --- |
| #include<iostream>  #include<iomanip>  #include<fstream>  #include<cstdlib>  //デバッグ時のディレクティブ定義  #define \_DEBUG  using namespace std;  class Food {  private:  string name; //食品名  int number; //個数  public:  Food(); //コンストラクタ  Food(string, int); //nameとnumberを設定可能なコンストラクタ  void printFood(); //nameとnumberを出力する  void increment(); //numberを1増やす  void decrement(); //numberを1減らす  bool check\_name(string);//食品名が一緒かどうかチェックする  string get\_name() { return name; }  int get\_number() { return number; }  };  Food::Food() {//コンストラクタ(食品名は空文字列""に、個数は0に初期化)  name = "";  number = 0;  }  Food::Food(string str, int u) {//コンストラクタ(食品名はstrに、個数はuに初期化)  name = str;  number = u;  }  void Food::printFood() {  cout << name << ":" << number << endl;  }  void Food::increment() {  number++;  }  void Food::decrement() {  number--;  }  bool Food::check\_name(string str) {  if (name == str) {  return true;  }  else {  return false;  }  }  class Storage {  private:  Food foods[300];  int cnt;  public:  Storage() { cnt = 0; }  void input\_foods();  void output\_foods();  void printStorage();  void sort\_by\_number();  void sort\_by\_name();  void increase\_foods(int, string, int);  void reduction\_foods(int);  };  void Storage::input\_foods() {  //ファイルからの入力の準備  cout << "Input Foods." << endl;;  cout << "Filename? :";  string fname;  #ifdef \_DEBUG  fname = "foods\_input.txt";  #else  cin >> fname;  #endif // \_DEBUG  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  while (fin >> current\_name) {  int i;  for (i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].increment();  break;  }  }  if (i == cnt) {  increase\_foods(i, current\_name, 1);  }  }  }  void Storage::output\_foods() {  //ファイルからの入力の準備  cout << "Output Foods." << endl;;  cout << "Filename? :";  string fname;  #ifdef \_DEBUG  fname = "foods\_output.txt";  #else  cin >> fname;  #endif // \_DEBUG  ifstream fin(fname.c\_str());  if (!fin) {  cerr << "File Not Found" << endl;  exit(1);//異常終了  }  string current\_name;//ファイルから一行読み込んだ食品名  Food unexistFoods[300];  int unexistFoodsNum = 0;  while (fin >> current\_name) {  //一行ずつ読んで配列から該当食品を探しデクリメント  for (int i = 0; i < cnt; i++) {  if (foods[i].check\_name(current\_name)) {  foods[i].decrement();  //食品の在庫が0になったらリストから削除  if (foods[i].get\_number() == 0)  　　reduction\_foods(i);  break;  }  //取り出そうとした食品がリストに存在しなかった場合の処理  //在庫がないのに取り出そうとした個数をカウントして配列に所持  if (i == cnt-1) {  if (unexistFoodsNum == 0) {  unexistFoods[0] = Food(current\_name, 1);  unexistFoodsNum++;  } else {  for (int i = 0; i < unexistFoodsNum; i++) {  if (unexistFoods[i].check\_name(current\_name)) {  unexistFoods[i].increment();  break;  }  if (i == unexistFoodsNum - 1) {  unexistFoods[unexistFoodsNum] = Food(current\_name, 1);  unexistFoodsNum++;  break;  }  }  }  }  }  }  //存在しないのに取り出そうしていた場合は足りていなかった食品の情報を出力  if (unexistFoodsNum > 0) {  cout << endl;  cout << "=====存在しないのに取り出そうとした食品=====" << endl;  for (int i = 0; i < unexistFoodsNum; i++) {  cout << "[" << setw(2) << i + 1 << "] ";  unexistFoods[i].printFood();  }  cout << "============================================" << endl;  }  }  void Storage::increase\_foods(int index, string foodName, int num) {  foods[index] = Food(foodName, num);  cnt++;  }  //foods配列から引数で指定した要素番号の要素を削除する  void Storage::reduction\_foods(int index) {  //削除する要素以降の要素を一つ後の要素で上書きして要素を詰める  for (int i = index; i < cnt - 1; i++) {  //次の要素ので上書き  swap(foods[i], foods[i + 1]);  }  cnt--;  }  void Storage::printStorage() {  //食品リストの表示  cout << endl;  cout << "////////////// All Food List ///////////////" << endl;  for (int i = 0; i < cnt; i++) {  cout << "[" << setw(2) << i + 1 << "] ";  foods[i].printFood();  }  cout << "////////////// All Food List END ///////////////" << endl;  cout << endl;  }  void Storage::sort\_by\_number()  {  for (int i = 0; i < cnt - 1; i++) {  for (int j = i + 1; j < cnt; j++) {  //食品の数が昇順になっていなければswapFlagをtrueにする  bool swapFlag = foods[j].get\_number() < foods[i].get\_number();  if (!swapFlag) continue;  //入れ替え  swap(foods[i], foods[j]);  }  }  }  void Storage::sort\_by\_name()  {  for (int i = 0; i < cnt - 1; i++) {  for (int j = i + 1; j < cnt; j++) {  //アルファベット順になっていなければswapFlagをtrueにする  bool swapFlag = foods[j].get\_name() < foods[i].get\_name();  if (!swapFlag) continue;  //入れ替え  swap(foods[i], foods[j]);  }  }  }  int main() {  Storage s1;  s1.input\_foods();  s1.printStorage();  s1.output\_foods();  s1.sort\_by\_name();  s1.printStorage();  } |

# 実行結果

実行結果と、挙動確認の為に追記したfoods\_output.txtファイルを以下掲載します。

実行結果 1 report01\_D04.cppの実行結果

|  |
| --- |
| Input Foods.  Filename? :foods\_input.txt  ////////////// All Food List ///////////////  [ 1] bread:6  [ 2] potato:2  [ 3] tomato:5  [ 4] apple:3  [ 5] milk:8  [ 6] rice:2  [ 7] butter:1  [ 8] orange:3  [ 9] cheese:1  ////////////// All Food List END ///////////////  Output Foods.  Filename? :foods\_output.txt  =====存在しないのに取り出そうとした食品=====  [ 1] milk:2  [ 2] bread:2  ============================================  ////////////// All Food List ///////////////  [ 1] apple:2  [ 2] cheese:1  [ 3] potato:1  [ 4] rice:2  [ 5] tomato:5  ////////////// All Food List END /////////////// |

2019/05/03平成31年5月3日

ファイル 1 foods\_output .txt

|  |
| --- |
| bread  orange  potato  apple  milk  bread  butter  milk  orange  milk  orange  milk  milk  milk  bread  bread  milk  milk  milk  milk  bread  bread  bread  bread |