

プログラミング演習II レポート

課題名: 課題1 文字列とポインタ

学籍番号	212273B
氏名	高木 竜哲
提出日	令和 5 年 5 月 11 日
書式修正版提出日	令和 5 年 月 日

【チェック項目】

以下の項目が正しく記載されているか確認し、○をつけること。

※ がついているものは必須項目ではない。

項目	自己 チェック
表紙に必要事項を記入したか？	<input type="radio"/>
章立てを行い、見易い構成となっているか？	<input type="radio"/>
課題 1～3 についてフローチャート等と本文によって、アルゴリズムの実現方法を説明したか？	<input type="radio"/>
ソースリストを載せたか？ソースリストの書式は見やすいか？ ソース中に適宜コメントを記し、説明がされているか？ (注意) プログラムでは適切にインデント(字下げ)を行い、空行や空白を適宜入れて見やすい形に整えること。	<input type="radio"/>
課題での使用を禁止されたライブラリ関数を使っていないか？	<input type="radio"/>
ソースリストについて説明したか？ (注意) ソースリスト中のコメントではなく本文中で説明すること。	<input type="radio"/>
実行結果を載せたか？ (注意) すべての実行について実行結果の画面ダンプを載せること。	<input type="radio"/>
実行結果について説明したか？ (注意) 入力として何を与え、どういう結果が出たのか？そして、それは意図した結果であるのか等を記述する。	<input type="radio"/>
考察を記述したか？（考察は感想ではない）	<input type="radio"/>
図表番号とタイトルをつけたか？図表番号は本文中で参照されているか？ (注意) 図のタイトルは図の下、表のタイトルは表の上につけること。通し番号を付け、タイトルは内容を適切に表したものにする。	<input type="radio"/>
参考文献を1本以上挙げ、本文中で参照したか？	<input type="radio"/>
ページ番号をつけたか？	<input type="radio"/>
※ いずれかのオプション課題を実現したか？	<input type="radio"/>
※ オプション課題として取り組んだ内容について説明したか？ プログラムが完成していなくても、検討したことがあれば記すこと。	<input type="radio"/>
※ オプション課題の実現方法を詳しく説明したか？	<input type="radio"/>

1. 課題 1 の目的

練習問題・課題問題を解くことにより、C 言語における文字型データと文字列データ、構造体、ポインタの基礎について学ぶ。

2. 課題 1 の概要

文字列の操作・構造体を用いた四則演算の操作・ポインタを用いた文字列配列の操作を行う。これまで今回学ぶことを使わずともできていたことを別の解法として学んでいく。

3. 課題 1-1

3.1 課題 1-1 の目的

文字列の変換・検索をできるようになる。

3.2 課題 1-1 の概要

与えられた文字列の中に特定の文字列が含まれているかどうか検索するプログラムを作成する。

3.3 データ構造

3.3.1. 定数宣言

MAXSTRLEN: 入力文字列の最大長 256

3.3.2. 型宣言

なし

3.3.3. グローバル変数

なし

3.4 アルゴリズムの説明

プログラム全体のフローチャートを図 1 に示す。

このプログラムでは、まず元の文字列・検索したい文字列を入力させ、それぞれ `str`・`pat` に格納し、表示する。その後文字列検索に入る。`pos` という文字列検索の現在位置が文字数より小さい間ループし、内部で `search_string` 関数を用いて検索する。見つかったら発見位置を表示する。ループを抜けた後、見つかった回数を出力し終了する。

`search_string` 関数はまず文字列を 2 つとも小文字に変換する。その後、文字列の先頭から順に検索したい文字列と比較していき、一致した場合先頭の位置を返して終了する。一致しなかった場合、-1 を返して終了する。

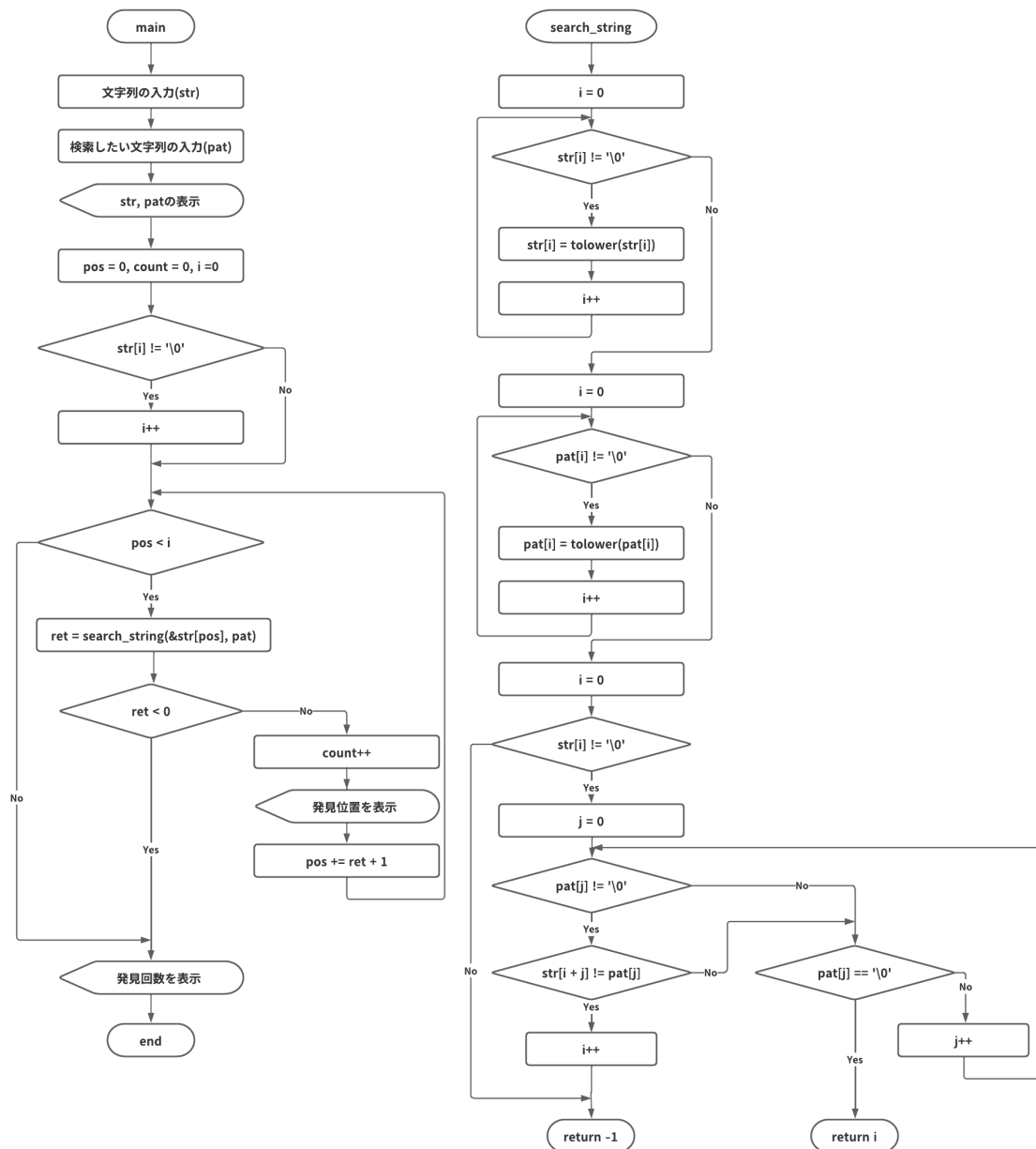


図 1 課題 1-1 のフローチャート (main, search_string)

3.5 ソースリスト

作成したプログラムのソースリストを以下のリスト 1 に示す。必要な部分に関してはコメントを記してある。

リスト 1 ある文字列に文字列が含まれるか検索するプログラム

```

1:  #include <stdio.h>
2:  #include <string.h>
3:  #include <ctype.h>
4:
5:  #define MAXSTRLEN 256 /* 入力文字列の最大長 */
6:
7:  /*
8:   * search_string:
  
```

```

 9:      *   文字列 'str'中に含まれる文字列 'pat'を探す
10:      *   大文字小文字を区別しない。
11:      *   文字列が含まれていればその場所を返す (先頭を 0 として何文字目か)
12:      *   含まれていなければ -1 を返す
13:      *
14:      */
15:  int search_string(char str[], char pat[])
16:  {
17:      int i, j, k;
18:
19:      /* 文字列の大文字を小文字に変換 */
20:      i = 0;
21:      while (str[i] != '\0')
22:      {
23:          str[i] = tolower(str[i]);
24:          i++;
25:      }
26:      i = 0;
27:      while (pat[i] != '\0')
28:      {
29:          pat[i] = tolower(pat[i]);
30:          i++;
31:      }
32:
33:      /* 文字列検索 */
34:      i = 0;
35:      while (str[i] != '\0')
36:      {
37:          /* 文字列の先頭から順に文字列を比較 */
38:          j = 0;
39:          while (pat[j] != '\0')
40:          {
41:              if (str[i + j] != pat[j])
42:              {
43:                  break;
44:              }
45:              j++;
46:          }
47:          /* 一致した文字数と pat の文字数が一致したら文字列が含まれているので、その位置を
返す */
48:          if (pat[j] == '\0')
49:          {
50:              return i;
51:          }
52:          i++;
53:      }
54:      return -1;
55:  }
56:
57:  int main(void)
58:  {
59:      int pos;          /* 文字列検索の現在位置 */
60:      int i;            /* ループカウンタ */
61:      char str[MAXSTRLEN]; /* 文字列入力用 (被検索文字列) */
62:      char pat[MAXSTRLEN]; /* 文字列入力用 (検索文字列) */
63:
64:      /* 2つの文字列を入力 */
65:      printf("input string = ");
66:      gets(str);
67:      printf("input pattern = ");
68:      gets(pat);
69:
70:      /* 入力した2つの文字列を表示 */
71:      printf("string=%s\n", str);
72:      printf("pattern=%s\n", pat);
73:
74:      /* 文字列検索 */
75:      pos = 0;
76:      int count = 0; // 見つかった回数を記録 [オプション課題]

```

```

77:         i = 0;
78:         while (str[i] != '\0')
79:         {
80:             i++;
81:         }
82:         while (pos < i)
83:         {
84:             int ret;
85:             ret = search_string(&str[pos], pat);
86:
87:             if (ret < 0) /* -1 が返ってきたらループを抜ける */
88:                 break;
89:             else
90:                 count++;
91:
92:             /* 文字列一致個所を発見 (str の pos からの相対位置が返る) */
93:             /* str の先頭からの位置は 'pos + ret' になるのでその値を表示する */
94:             printf("found at %d¥n", pos + ret);
95:
96:             pos += ret + 1; /* 検索位置を進める */
97:         }
98:         /* 見つかった回数を出力 [オプション課題] */
99:         printf("¥nfound %d time(s)¥n", count);
100:
101:         return 0;
102:     }

```

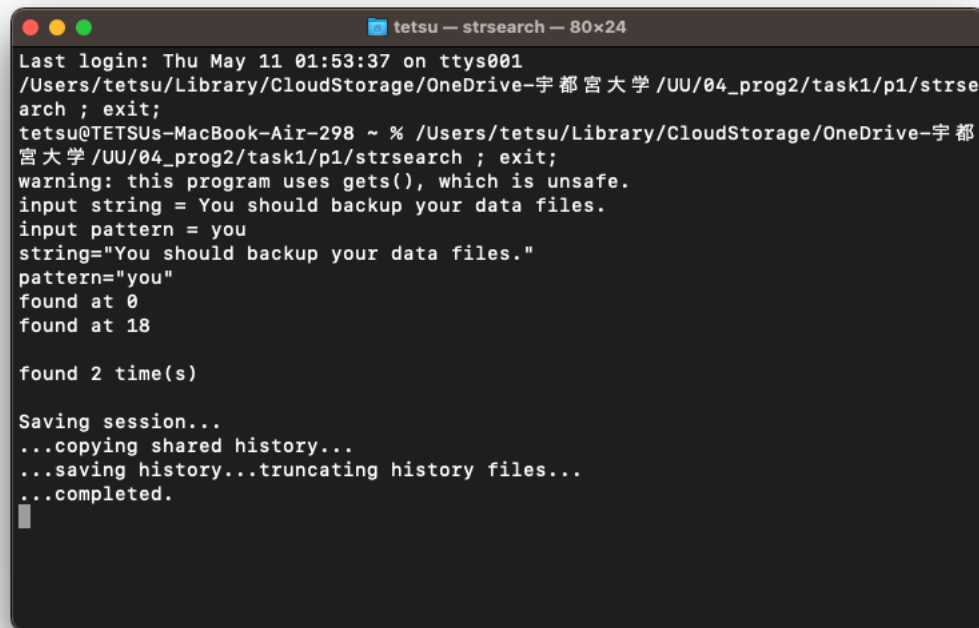
このコードは与えられた文字列内で指定された文字列を検索するプログラムである。5 行目では入力文字列の最大長を示す定数 **MAXSTRLEN** を 256 で定義している。15 行目から 55 行目までは **search_string** 関数として、文字列 **str** 中に文字列 **pat** を探し、見つかった場所を返すプログラムを定義している。65 行目から 68 行目ではユーザーに対して文字列の入力を促すプロンプトが表示され、71 行目から 72 行目では入力された文字列 **str** と **pat** が表示される。94 行目では見つかった文字列の位置が表示される。

このプログラムにおいては **strlen** 関数が使えなかったため、文字型変数の最後には **'\0'** が加えられ、**while** 文で判断し代用した。

3.6 実行結果

“You should backup your data files.” という文字列を与え、“you”を検索したい文字列として与えた場合の実行結果が以下の図 2 である。正しくコーディングできていれば大文字小文字を問わず検索できるため、文頭の “You” と 4 単語目最初に存在する “you” の 2 つが出力されると予想した。

“You” と “you”、どちらも正しく認識し、文字の位置を出力していることがわかる。そして文字列中に 2 回出現したため、その旨もきちんと表示できていることがわかる。よって実験前の予想と等しくなり、正しくコーディングできていることがわかった。



```
tetsu — strsearch — 80x24
Last login: Thu May 11 01:53:37 on ttys001
/Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学/UU/04_prog2/task1/p1/strsearch ; exit;
tetsu@TETSUs-MacBook-Air-298 ~ % /Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学/UU/04_prog2/task1/p1/strsearch ; exit;
warning: this program uses gets(), which is unsafe.
input string = You should backup your data files.
input pattern = you
string="You should backup your data files."
pattern="you"
found at 0
found at 18

found 2 time(s)

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
```

図 2 課題 1-1 プログラムの実行結果

3.7 考察

今回のプログラムは分岐が多く、条件指定が難しかったが比較的容易に実装できた。しかし、`search_string` 関数の中身がやや単調であると感じたため、文字列配列の 2 次元配列を用いて実装するなど、工夫できる部分があるのではないかと考えた。

3.8 オプション課題

検索文字列の発見回数を表示している。これは元のプログラムに新たにカウント変数 `count` を追加することによって実現した。リスト 1 のソースコードにおいて、76・98 行目を追加しており、結果は図 2 のように正しく出力されることがわかった。

4. 課題 1-2

4.1 目的

構造体を理解し、分数による四則演算を行う。

4.2 概要

構造体を用いて分数を扱う四則演算を行うプログラムを作成する。この際、負の値になるならば分子側にマイナスを表示する。また、既約分数として表示するため、ユークリッドの互除法等を利用する。

4.3 データ構造

4.3.1. 定数宣言

なし

4.3.2. 型宣言

`rational_number rational_number_add(rational_number a, rational_number b)`

: 分数である有理数の加算(2つの引数で与えられた分数の和を求めて返す)

`rational_number rational_number_sub(rational_number a, rational_number b)`

: 分数である有理数の減算(2つの引数で与えられた分数の差を求めて返す)

`rational_number rational_number_mul(rational_number a, rational_number b)`

: 分数である有理数の乗算(2つの引数で与えられた分数の積を求めて返す)

`rational_number rational_number_div(rational_number a, rational_number b)`

: 分数である有理数の除算(2つの引数で与えられた分数の商を求めて返す)

4.3.3. グローバル変数

なし

4.4 アルゴリズムの説明

プログラムのメイン部分・ユークリッドの互除法部分・分数の既約化部分のフローチャートを以下の図3に、四則演算の計算部分を以下の図4に示す。

このプログラムではまずメイン関数内で4つの数字を入力させる(1つ目の分数の分子・分母、2つ目の分数の分子・分母)。その後、`test_rational_number_calc` 関数を用いて内部で四則演算関数を呼び出し、計算結果を出力する。

四則演算関数では `result` という構造体を定義した後、`result.numer`, `result.demon` それぞれについて計算を行い、`result` を返す。

ユークリッドの互除法関数 (`gcd`) では、引数として与えられた2数について、互いに割った際の余りが0であれば小さい方の数を返し、そうでなければ小さい方の数と余りの値で再度 `gcd` 関数を実行する。このように再帰的に関数を呼び出しユークリッドの互除法を実行する。

分数の既約化関数 (`rational_number_reduced`) では、引数として与えられた分子・分母の値に関してまずユークリッドの互除法を実行し、返ってきた値(最大公約数)が0でなければその値で分子・分母を割る。また、分母が負の値であれば分子・分母それぞれに-1を掛けることで、負の値の際は必ず分子に負の符号(マイナス)がつくようにしている。

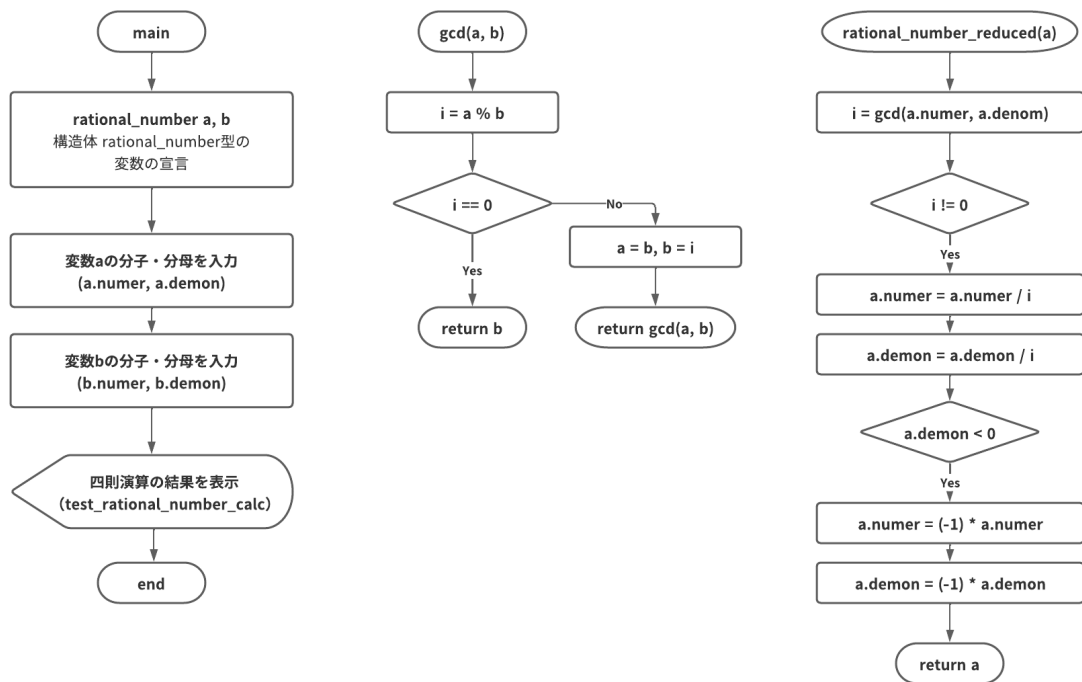


図 3 課題 1-2 のフローチャート (main, gcd, rerational_number_reduced)

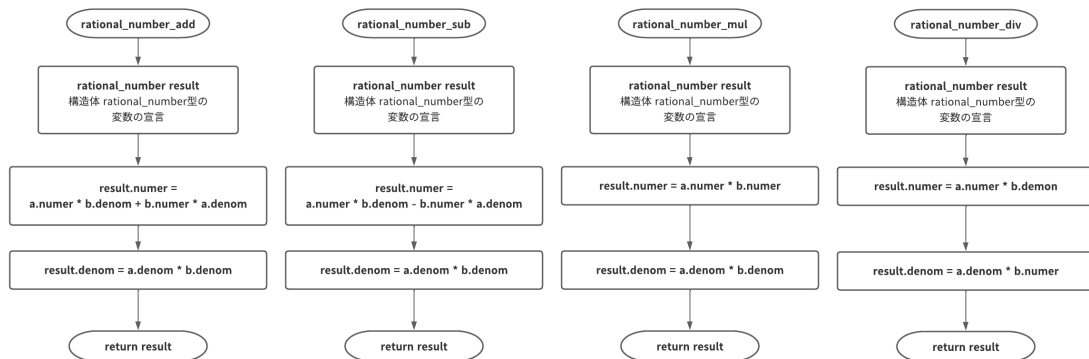


図 4 課題 1-2 のフローチャート (四則演算部分)

4.5 ソースリスト

作成したプログラムのソースリストを以下のリスト 2 に示す。必要な部分に関してはコメントを記してある。

リスト 2 分数における四則演算を行うプログラム

```

1:  /*
2:   * rat-num.c
3:   * rational number calculation program
4:   */
5:  #include <stdio.h>
6:
7:  /* 有理数(分数)を表現する構造体 */
8:  typedef struct rational_number
9:  {
10:     int numer; /* 分子 (numerator) */
11:     int denom; /* 分母 (denominator) */
12:  } rational_number;
13:

```



```

14:  /* 有理数(分数)の加算 */
15:  /* 2つの引数で与えられた分数の和を求めて返す */
16:  rational_number
17:  rational_number_add(rational_number a, rational_number b)
18:  {
19:      rational_number result;
20:
21:      result.number = a.number * b.denom + b.number * a.denom;
22:      result.denom = a.denom * b.denom;
23:
24:      return result;
25:  }
26:
27:  /* 有理数(分数)の減算 */
28:  /* 2つの引数で与えられた分数の差を求めて返す */
29:  rational_number
30:  rational_number_sub(rational_number a, rational_number b)
31:  {
32:      rational_number result;
33:
34:      result.number = a.number * b.denom - b.number * a.denom;
35:      result.denom = a.denom * b.denom;
36:
37:      return result;
38:  }
39:
40:  /* 有理数(分数)の乗算 */
41:  /* 2つの引数で与えられた分数の積を求めて返す */
42:  rational_number
43:  rational_number_mul(rational_number a, rational_number b)
44:  {
45:      rational_number result;
46:
47:      result.number = a.number * b.number;
48:      result.denom = a.denom * b.denom;
49:
50:      return result;
51:  }
52:
53:  /* 有理数(分数)の除算 */
54:  /* 2つの引数で与えられた分数の商を求めて返す */
55:  rational_number
56:  rational_number_div(rational_number a, rational_number b)
57:  {
58:      rational_number result;
59:
60:      result.number = a.number * b.denom;
61:      result.denom = a.denom * b.number;
62:
63:      return result;
64:  }
65:
66:  /* ユークリッドの互除法 */
67:  int gcd(int a, int b)
68:  {
69:      int i;
70:
71:      i = a % b;
72:      if (i == 0)
73:      {
74:          return b;
75:      }
76:      else
77:      {
78:          a = b;
79:          b = i;
80:          return gcd(a, b);
81:      }
82:  }
83:

```

```

84:    /* 有理数(分数)の既約化 */
85:    /* 引数で与えられた分数の既約分数を求めて返す */
86:    rational_number
87:    rational_number_reduced(rational_number a)
88:    {
89:        int i = gcd(a.number, a.denom);
90:
91:        if (i != 0)
92:        {
93:            a.number = a.number / i;
94:            a.denom = a.denom / i;
95:        }
96:
97:        if (a.denom < 0)
98:        {
99:            a.number = (-1) * a.number;
100:           a.denom = (-1) * a.denom;
101:        }
102:
103:        return a;
104:    }
105:
106:    /* 有理数(分数)の表示 */
107:    void rational_number_print(rational_number r)
108:    {
109:        /*
110:         * "[ 分子/分母 ]" の形式で表示
111:         *
112:         * ただし、
113:         * 0 のときはそのまま "0" と表示
114:         * 1 のときはそのまま "1" と表示
115:         */
116:
117:        r = rational_number_reduced(r); /* 表示前に既約化 */
118:
119:        if (r.number == 0)
120:            printf("0");
121:        else if (r.number == 1 && r.denom == 1)
122:            printf("1");
123:        else
124:            printf("[ %d/%d ]", r.number, r.denom);
125:    }
126:
127:    /* ここから先は動作テスト用の関数 */
128:
129:    void test_rational_number_calc(rational_number a, rational_number b, char
optype)
130:    {
131:        rational_number c;
132:
133:        switch (optype)
134:        {
135:            case '+':
136:                c = rational_number_add(a, b);
137:                break;
138:            case '-':
139:                c = rational_number_sub(a, b);
140:                break;
141:            case '*':
142:                c = rational_number_mul(a, b);
143:                break;
144:            case '/':
145:                c = rational_number_div(a, b);
146:                break;
147:            default:
148:                printf("unknown operation type %c\n", optype);
149:                return;
150:                break;
151:        }
152:

```

```

153:      /* 計算内容の表示 */
154:      rational_number_print(a);
155:      printf(" %c ", optype);
156:      rational_number_print(b);
157:      printf(" = ");
158:      rational_number_print(c);
159:      printf("¥n");
160:  }
161:
162:  int main(void)
163:  {
164:      rational_number a, b; /* 構造体 rational_number 型の変数の宣言 */
165:
166:      /* キーボードから 4 つの数を入力 */
167:      printf("input number (a.numer) = ");
168:      scanf("%d", &a.numer);
169:      printf("input number (a.denom) = ");
170:      scanf("%d", &a.denom);
171:      printf("input number (b.numer) = ");
172:      scanf("%d", &b.numer);
173:      printf("input number (b.denom) = ");
174:      scanf("%d", &b.denom);
175:
176:      /* 2 つの有理数(分数) の四則演算結果を表示 */
177:      test_rational_number_calc(a, b, '+');
178:      test_rational_number_calc(a, b, '-');
179:      test_rational_number_calc(a, b, '*');
180:      test_rational_number_calc(a, b, '/');
181:
182:      return 0;
183:  }

```

これは有理数（分数）の四則演算を行うプログラムである。8 行目から 12 行目では有理数を表現するための構造体 **rational_number** が定義され、15 行目から 25 行目では加算を行う関数が、28 行目から 38 行目では減算を行う関数が、41 行目から 51 行目では乗算を行う関数が、54 行目から 64 行目では除算を行う関数がそれぞれ定義されている。これらは 2 つの分数を引数としてとり、関数内で定義する **result** という構造体に結果を記録しこの値を返す。67 行目から 82 行目では最大公約数を求める関数 **gcd** が、85 行目から 104 行目では既約分数に変換する関数 **rational_number_reduced** が、107 行目から 125 行目では有理数を表示する関数 **rational_number_print** がそれぞれ定義されている。**rational_number_print** において必要なのが **rational_number_reduced** であり、これに必要なのが **gcd** である。このように定義した関数を用いて **main** 関数内で 2 つの分数をユーザーに入力してもらい、その計算を行っている。

4.6 実行結果

以下に(a, b) = (10/20, 30/40), (3/5, 7/9), (4/6, 10/12)を入力し実行した結果をそれぞれ図 5、図 6、図 7 に示す。

```
tetsu — rat-num — 80x24
Last login: Thu May 11 05:47:17 on ttys004
/Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学 /UU/04_prog2/task1/p2/rat-num ; exit;
tetsu@TETSUs-MacBook-Air-298 ~ % /Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学 /UU/04_prog2/task1/p2/rat-num ; exit;
input number (a.number) = 10
input number (a.denom) = 20
input number (b.number) = 30
input number (b.denom) = 40
[ 1/2 ] + [ 3/4 ] = [ 5/4 ]
[ 1/2 ] - [ 3/4 ] = [ -1/4 ]
[ 1/2 ] * [ 3/4 ] = [ 3/8 ]
[ 1/2 ] / [ 3/4 ] = [ 2/3 ]

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
```

図 5 分数の四則演算結果 (10/20, 30/40)

```
tetsu — rat-num — 80x24
Last login: Thu May 11 09:12:49 on ttys004
/Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学 /UU/04_prog2/task1/p2/rat-num ; exit;
tetsu@TETSUs-MacBook-Air-298 ~ % /Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学 /UU/04_prog2/task1/p2/rat-num ; exit;
input number (a.number) = 3
input number (a.denom) = 5
input number (b.number) = 7
input number (b.denom) = 9
[ 3/5 ] + [ 7/9 ] = [ 62/45 ]
[ 3/5 ] - [ 7/9 ] = [ -8/45 ]
[ 3/5 ] * [ 7/9 ] = [ 7/15 ]
[ 3/5 ] / [ 7/9 ] = [ 27/35 ]

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

図 6 分数の四則演算結果 (3/5, 7/9)

```
tetsu — rat-num — 80x24
Last login: Thu May 11 09:12:59 on ttys004
/Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学 /UU/04_prog2/task1/p2/rat-num ; exit;
tetsu@TETSUs-MacBook-Air-298 ~ % /Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学 /UU/04_prog2/task1/p2/rat-num ; exit;
input number (a.number) = 4
input number (a.denom) = 6
input number (b.number) = 10
input number (b.denom) = 12
[ 2/3 ] + [ 5/6 ] = [ 3/2 ]
[ 2/3 ] - [ 5/6 ] = [ -1/6 ]
[ 2/3 ] * [ 5/6 ] = [ 5/9 ]
[ 2/3 ] / [ 5/6 ] = [ 4/5 ]

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

図 7 分数の四則演算結果 (4/6, 10/12)

- $(a, b) = (10/20, 30/40)$ について、
 $a+b = 50/40 = 5/4$, $a-b = -10/20 = -1/2$, $a*b = 300/800 = 3/8$, $a/b = 400/600 = 2/3$
であり、図 5 にはこの値が正しく表示されている。
- $(a, b) = (3/5, 7/9)$ について、
 $a+b = 62/45$, $a-b = -8/45$, $a*b = 21/45 = 7/15$, $a/b = 27/35$
であり、図 6 にはこの値が正しく表示されている。
- $(a, b) = (4/6, 10/12)$ について、
 $a+b = 18/12 = 3/2$, $a-b = -2/12 = -1/6$, $a*b = 40/72 = 5/9$, $a/b = 12/15 = 4/5$
であり、図 7 にはこの値が正しく表示されている。

これら 3 例の実行により、このプログラムは正しく動作していると分かる。

4.7 考察

今回のプログラムは今までと比べて関数の数がかかなり多かった。

test_rational_number_calc 関数内で case 関数を用いて場合分けしているのであれば、この関数内だけで四則演算ができるように考えられたのでぜひ一度試行してみたい。

5. 課題 1-3

5.1 目的

ポインタを用いた文字列の操作を学ぶ。

5.2 概要

ポインタで文字列の配列を扱い、入力した文字列の文字数を表示した上で反転させ、入力された文字列と反転させた文字列の任意の位置の文字を表示させるプログラムを作成する。この際、配列を用いた処理ではなくポインタを用いた処理で実現させる。

5.3 データ構造

5.3.1. 定数宣言

なし

5.3.2. 型宣言

なし

5.3.3. グローバル変数

なし

5.4 アルゴリズムの説明

プログラム全体のフローチャートを図 8 に示す。

これは入力した文字列の長さ・反転した文字列を出力し、その後入力された文字の位置に応じてもとの文字列・反転後の文字列のその位置の値を出力するプログラムである。文字列の反転に関してはポインタを用いて行っている。

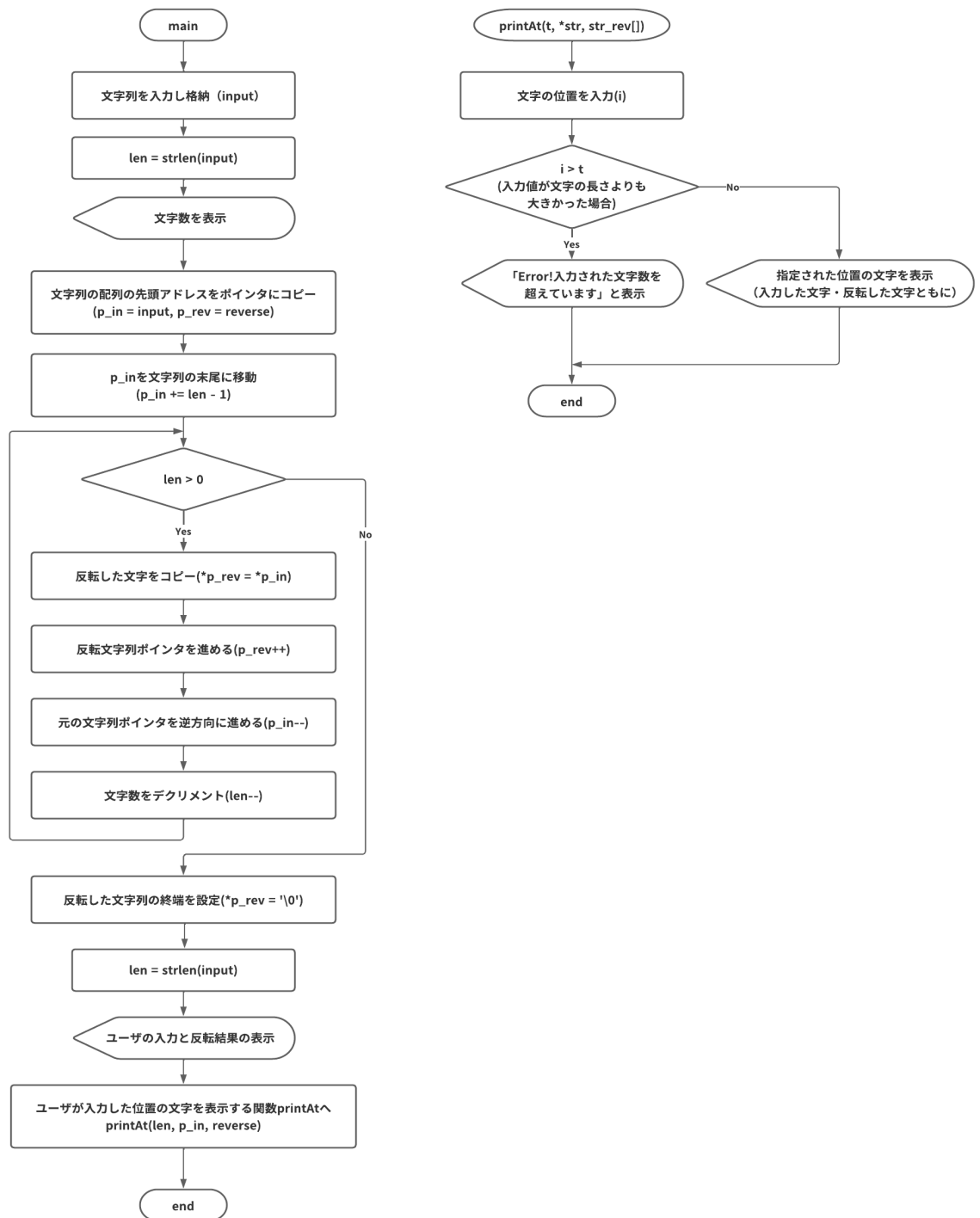


図 8 課題 1-3 のフローチャート

5.5 ソースリスト

作成したプログラムのソースリストを以下のリスト 3 に示す。必要な部分に関してはコメントを記してある。

リスト 3 分数における四則演算を行うプログラム

```

1:  #include <stdio.h>
2:  #include <string.h>
3:
4:  void printAt(int t, char *str, char str_rev[])
5:  {
6:
7:      /*
8:      3-2:この関数の中身を作る
9:      */
10:     int i;
11:     printf("文字の位置を入力してください> ");
12:     scanf("%d", &i);
13:     printf("ユーザの入力: %d\n", i);
14:     /*ユーザの入力が文字数を超えていた場合の処理*/
15:     // printf("文字数は%d 文字\n", t);
16:     if (i > t)
17:     {
18:         printf("Error!入力された文字数を超えています\n");
19:     }
20:     else
21:     {
22:         /*それ以外の場合は指定された位置の文字を表示*/
23:         printf("ユーザが入力した文字列の場合: %c\n", str[i]);
24:         printf("反転した文字列の場合: %c\n", str_rev[i - 1]);
25:     }
26: }
27:
28: int main(void)
29: {
30:     char input[20], *p_in, reverse[20], *p_rev;
31:     int len = 0; /*文字数を格納する変数*/
32:
33:     printf("文字列を入力してください\n");
34:     scanf("%s", input);
35:
36:     /*
37:     3-1:ここに入力された文字数を把握するための処理を追加
38:     */
39:     len = strlen(input);
40:
41:     printf("文字数は%d 文字\n", len); /*文字数表示*/
42:
43:     /*文字列の配列の先頭アドレスをポインタにコピー*/
44:     p_in = input;
45:     p_rev = reverse;
46:
47:     /*
48:     3-1:ここに文字列反転の処理を作る
49:     */
50:     p_in += len - 1; // p_in を文字列の末尾に移動
51:     while (len > 0)
52:     {
53:         *p_rev = *p_in; // 反転した文字をコピー
54:         p_rev++;        // 反転文字列ポインタを進める
55:         p_in--;         // 元の文字列ポインタを逆方向に進める
56:         len--;          // 文字数をデクリメント
57:     }
58:     *p_rev = '\0'; // 反転した文字列の終端を設定
59:
60:     // 文字数を再度取得
61:     len = strlen(input);
62:
63:     /*ユーザの入力と反転結果の表示*/
64:     printf("ユーザの入力: %s\n", input);
65:     printf("反転した出力: %s\n", reverse);
66:

```



```

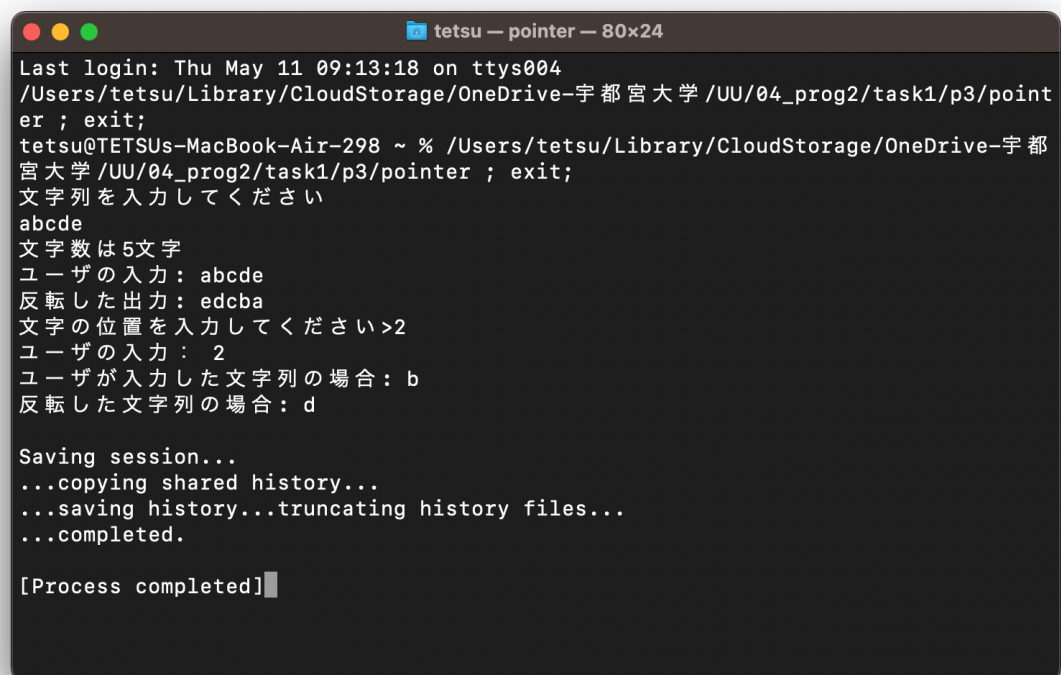
67:      /*ユーザが入力した位置の文字を表示する関数 printAt へ*/
68:      printAt(len, p_in, reverse); /*ユーザが入力した文字列はポインタ, 反転した文字
列は配列を渡す*/
69:
70:      return 0;
71:  }

```

これはユーザーが入力した文字列を反転し、反転前後における指定の位置の文字を出力するプログラムである。4 行目から 26 行目では、ユーザーから文字の位置を入力してもらい、正常な値であれば指定位置の文字を表示する関数 `printAt` が定義されている。`main` 関数においてはユーザーからの入力を受け取り、文字列の反転と指定位置の文字表示を行う。具体的には 30 行目から 42 行目においてユーザからの文字列入力を受け取り、文字列の長さを取得する。50 行目から 57 行目において、文字列配列のポインタを用いて文字列を反転させる。その後、元の文字列と反転した文字列を表示し、`printAt` 関数を呼び出すことでユーザが入力した位置の文字を表示している。

5.6 実行結果

「abcde」、「ittetsutakaki(自身の名前)」を入力し実行した場合の実行結果をそれぞれ以下の図 9、図 10 に示す。



```

tetsu — pointer — 80x24
Last login: Thu May 11 09:13:18 on ttys004
/Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学/UU/04_prog2/task1/p3/pointer ; exit;
tetsu@TETSUs-MacBook-Air-298 ~ % /Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学/UU/04_prog2/task1/p3/pointer ; exit;
文字列を入力してください
abcde
文字数は5文字
ユーザの入力: abcde
反転した出力: edcba
文字の位置を入力してください>2
ユーザの入力: 2
ユーザが入力した文字列の場合: b
反転した文字列の場合: d

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]

```

図 9 課題 1-3 実行結果 (abcde)

```
tetsu — pointer — 80x24
Last login: Thu May 11 09:52:31 on ttys004
/Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学/UU/04_prog2/task1/p3/pointer ; exit;
tetsu@TETSUs-MacBook-Air-298 ~ % /Users/tetsu/Library/CloudStorage/OneDrive-宇都宮大学/UU/04_prog2/task1/p3/pointer ; exit;
文字列を入力してください
ittetsutakaki
文字数は13文字
ユーザの入力: ittetsutakaki
反転した出力: ikakatustetti
文字の位置を入力してください>3
ユーザの入力: 3
ユーザが入力した文字列の場合: t
反転した文字列の場合: a

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

図 10 課題 1-3 実行結果 (ittetsutakaki)

5 文字の文字列「abcde」を反転すると「edcba」であり、2 文字目はそれぞれ「b」、「e」である。図 9 ではこの通り出力されている。

13 文字の文字列「ittetsutakaki」を反転すると「ikakatustetti」であり、3 文字目はそれぞれ「t」、「a」である。図 10 ではこの通り出力されている。

これらの実行結果より、このプログラムは正しく動作していると分かる。

5.7 考察

ポインタを用いた文字列の処理について、一文字ずつ出力したり利用したりするのに適していると感じた[1]。また、書き換え処理を行うにはポインタを用いる必要があるため、理解を深めていきたいと考えた。

6. 課題 1 のまとめ

課題 1 を通して文字列データの扱い方について、課題 2 を通して構造体の扱い方について、課題 3 を通してポインタを用いた文字列の扱い方について学ぶことができた。特にポインタに関しては少々苦手分野であり、何度も課題 3 の実行結果に指定していない文字列が現れた。これはアドレスの間違いであり、特に何もしていない位置に含まれる値が出力されたものであった。今後もポインタ・文字列配列について理解を深めるため、自主学習に努めていきたいと思った。

[参考文献]

[1] [c 言語]文字列を 1 文字ずつ参照する方法, <https://it-ojisan.tokyo/c-str-loop/>, 2023/05/11 閲覧