プログラミング II 前期中間試験

準備　プログラムを作る前に、以下の操作をしてファイルの準備をしておくこと。

1. 授業の配付資料を全てダウンロードする場合は、以下を実行する（既に実行済みの場合は不要）
   `$ mygitclone-p2`
2. GitHub から自分のリポジトリを clone しておく（既に実行済みの場合は不要）
   `$ mygitclone 「自分の GitHub ユーザ名」`
   `$ cd prog3i-(ユーザ名)`
   `$ ./myconf`
3. 今回の定期試験用のフォルダをコピーする
   `$ cd ~/prog3i-(ユーザ名)`
   `$ cp -r /usr/local/common/kogai/p2/test1mid .` （←ここにピリオド）
   `$ cd test1mid` （コピーしたフォルダに移動する）
   `$ ls` （フォルダ内のファイルを確認すると、以下のファイルがコピーされている）
   `1_git.txt 3_list.c 5_arg.c`
   `2_fill.c 4_list2.c`
4. テキストエディタでプログラムを開き、**まず先頭行に C のコメントとして自分の番号と名前を書いて**、解答を始める
   `$ gedit` 各問のファイル名 `&`

1　GitHub を利用したバージョン管理において、次の（1）～（2）の説明に適した git コマンドを以下の語群（a）～（e）から選び、ファイル「1_git.txt」に解答しなさい。

（1）リポジトリの変更履歴が GitHub に反映される
（2）バージョン管理するファイルを追加する

```
(a) add   (b) clone   (c) commit   (d) config   (e) push
```

2　int 型の配列のためのメモリを確保し、仮引数で与えられた整数 num から 1 までの整数が降順で格納された配列を作る関数 fill_int() をファイル「2_fill.c」に作成しなさい。この関数のプロトタイプ宣言は以下のようになる。

```
int *fill_int(int num);
 /* num から 1 までの整数を格納するのに必要な配列の要素数分のメモリを確保する */
 /* 作成した配列の先頭から、num, num-1, num-2, ..., 1 の整数を格納していく */
 /* 作成した配列のアドレスを return で戻す */
 /* なお、「配列に格納する場所（?番目）」と「格納する値」は異なるため、別々の変数を用意した方が作りやすい */
```

main() での動作確認の例とその実行結果は以下のようになる。なお、ここで使っている出力用関数 main() は「2_fill.c」に用意されている。

```
[main() での処理]
int *a;
int i;
a = fill_int(5);      /* 処理その 1 */
for(i=0; i<5; i++) {
    printf("%d ", *(a+i));
}
printf("\n");
free(a);
a = fill_int(100);    /* 処理その 2 */
```

```
for(i=0; i<100; i++) {
    printf("%d ", *(a+i));
}
printf("\n");
free(a);
[実行結果]
5 4 3 2 1              (←処理その 1：5〜1 の整数が格納されている)
100 99 98 97 96 95 94 93 92  (長いので以下省略)    (←処理その 2：100〜1 の整数が格納されている)
```

3 次のような分数を表すリスト用の構造体を考える。この構造体の宣言は「3_list.c」に用意されている。

```
typedef struct Frac {
    int bunshi;
    int bunbo;
    struct Frac *next;
} Frac;
```

この構造体 Frac を使った線形リストに対して、「引数で指定した分子と分母からなる分数をリストに追加する」関数 add_frac() をファイル「3_list.c」に作成しなさい。この関数のプロトタイプ宣言は以下のようになる。

```
void add_frac(Frac *p, int s, int b);
 /* 仮引数 p が参照しているリストに新しい要素を追加する */
 /* 新しい領域を確保して、仮引数 s, b を、確保した構造体 Frac のメンバ bunshi, bunbo にそれぞれ格納する */
 /* 新しく作った要素は、第 5 週と同様にリストの先頭に追加する */
```

main() での動作確認の例とその実行結果は以下のようになる。なお、ここで使っている出力用関数 show() と main() は「3_list.c」に用意されている。

```
[main の処理]
Frac head1;
head1.bunshi = 0; head1.bunbo = 0;   /* 空のスタックを準備する */
head1.next = NULL;
show(&head1, "head1 (1)");           /* 初期状態を出力する */
add_frac(&head1, 3, 8);              /* 3 回追加する */
add_frac(&head1, 2, 9);
add_frac(&head1, 7, 2);
show(&head1, "head1 (2)");           /* 追加後の状態を出力する */
[実行結果]
--- head1 (1) ---
0/0
--- head1 (2) ---
0/0
7/2
2/9
3/8
```

4 前問で示した構造体 Frac を使った線形リストに対して、「引数で指定した分子と分母からなる分数が仮分数（分子が分母と同じか大きい分数）の場合は、逆数（分子と分母を入れ替えた数）をリストに追加する」関数 add_frac2() をファイル「4_list2.c」に作成しなさい。この関数のプロトタイプ宣言は以下のようになる。

```
void add_frac2(Frac *p, int s, int b);
 /* 仮引数 p が参照しているリストに新しい要素を追加する */
```

```
/* 新しい領域を確保した後に、仮引数 s, b を比較して、仮分数となる場合はメンバ bunbo, bunshi にそれぞれ格納する */
/* そうでない場合は、s, b をメンバ bunshi, bunbo にそれぞれ格納する */
/* 新しく作った要素は、第 5 週と同様にリストの先頭に追加する */
```

main() での動作確認の例とその実行結果は以下のようになる。なお、ここで使っている出力用関数 show() と main() は「4_list2.c」に用意されている。

```
[main の処理]
Frac head2;
head2.bunshi = 0; head2.bunbo = 0;   /* 空のスタックを準備する */
head2.next = NULL;
show(&head2, "head2 (1)");           /* 初期状態を出力する */
add_frac2(&head2, 4, 7);             /* 真分数を追加する */
add_frac2(&head2, 5, 2);             /* 仮分数を追加する */
add_frac2(&head2, 9, 9);             /* 仮分数を追加する */
show(&head2, "head2 (2)");           /* 追加後の状態を出力する */
[実行結果]
--- head2 (1) ---
0/0
--- head2 (2) ---
0/0
9/9
2/5
4/7
```

5 コマンドライン引数で指定した文字列に対して、「文字列の長さ（つまり文字数）が最大のものを求める」プログラムをファイル「5_arg.c」に作成しなさい。

【main の補足】
- 何番目の文字列が最長さなのかを残す変数を用意する（例えば x）
- 1 番目以降の argv の文字列に対して繰り返しながら、
  x 番目の文字列よりも i 番目の文字列の方が長い場合は、最長の場所を更新する
- 繰り返し後に、最も長い文字列を出力する

実行例とその実行結果は以下のようになる。

```
[実行例とその実行結果]
$ ./a.out one two three
最も長い文字列: three
$ ./a.out 1111 22 333 4
最も長い文字列: 1111
```

問題はここまで

（問 1：4 点, それ以外：各 24 点）

定期試験の実施について

試験中に使用できるもの

- 筆記用具（メモ用紙は必要な人に配布）
- 演習室のコンピュータ一台（一つの机に一人の配置で、座る場所はどこでもよい）

試験中に参照できるもの

- 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
  （定期試験では紙媒体のものは参照不可）

* **上記以外の情報を参照することは不正行為とする**
（例：USB で接続された機器に保存されているファイルの参照、Web ブラウザやネットワークを介した情報の参照、自分の PC を使用する、など）

* 試験中（開始 5 分後〜開始 60 分後）は、演習室外へのネットワークアクセスは遮断される

* GitHub への提出のためのコマンドに限ってネットワーク利用が可能（それ以外は不正行為とする）

答案の提出

1. **提出する全てのファイルの先頭行に、C のコメントとして自分の番号と名前を書く**
2. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
$ git commit -m "前期中間提出"
$ git push origin master
```

（push が成功すると「done」が含まれたメッセージが表示される）

3. 提出が完了しているかを確認したい人は声をかけて下さい。（その場で教員側の画面で確認します）

前期中間試験 模範解答（平均 71.7 点）

採点について コンパイル時にエラーとなる箇所は −4 点, 実行可能だか処理内容が問題の意図と違う箇所は −2 点を基本とする。

■問 1

以下に解答する
(1) e (push)
(2) a (add)

■問 2

```
#include <stdio.h>
#include <stdlib.h>

/* 関数のプロトタイプ宣言 */
int *fill_int(int num);

/* num〜1 の整数が格納された配列を作る */
int *fill_int(int num)
{
    int *array;
    int n, i;
    /* int 型のメモリを確保する */
    array = (int *)malloc(sizeof(int)*num);
    if(array==NULL) {
        printf("not allocated.\n");
        return NULL;
    }
    /* 確保する整数の初期値は s */
    n = num;
    /* 確保したメモリに s〜e の整数を入れる */
    for(i=0; i<num; i++) {
        *(array+i) = n;
        n--;
    }
    /* 確保したメモリの先頭アドレスを戻す */
    return array;
}

int main(void)
{
    /* fill_int() の動作確認 */
    int *a;
    int i;
    a = fill_int(5);
    for(i=0; i<5; i++) {
        printf("%d ", *(a+i));
    }
    printf("\n");
    free(a);
    a = fill_int(100);
    for(i=0; i<100; i++) {
        printf("%d ", *(a+i));
    }
    printf("\n");
    free(a);

    return 0;
}
```

■問 3

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Frac {
    int bunshi;
    int bunbo;
    struct Frac *next;
} Frac;

void add_frac(Frac *p, int s, int b);
void show(Frac *p, char *str);

/* リストを出力する */
void show(Frac *start, char *str)
{
    Frac *p;
    printf("--- %s ---\n", str);
    for(p = start; p!=NULL; p = p->next) {
        printf("%d/%d\n", p->bunshi, p->bunbo);
    }
}

/* リストに要素を追加する */
void add_frac(Frac *p, int s, int b)
{
    Frac *new;
    /* 構造体 Frac の領域を確保する */
    new = (Frac *)malloc(sizeof(Frac));
    /* 確保した領域のメンバに引数の値を代入する */
    new->bunshi = s;
    new->bunbo = b;
    /* 確保した領域の next を更新する */
    new->next = p->next;
    /* p の next は確保した領域を参照する */
    p->next = new;
}

int main(void)
{
    /* add_frac() の動作確認 */
    Frac head1;
    head1.bunshi = 0; head1.bunbo = 0;
    head1.next = NULL;
    show(&head1, "head1 (1)");
    add_frac(&head1, 3, 8);
    add_frac(&head1, 2, 9);
    add_frac(&head1, 7, 2);
    show(&head1, "head1 (2)");

    return 0;
}
```

■問 4

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Frac {
    int bunshi;
    int bunbo;
    struct Frac *next;
} Frac;

void add_frac2(Frac *p, int s, int b);
void show(Frac *p, char *str);
```

```
/* リストを出力する */
void show(Frac *start, char *str)
{
    Frac *p;
    printf("--- %s ---\n", str);
    for(p = start; p!=NULL; p = p->next) {
        printf("%d/%d\n", p->bunshi, p->bunbo);
    }
}

/* リストに要素を追加する（その 2） */
void add_frac2(Frac *p, int s, int b)
{
    Frac *new;
    /* 構造体 Frac の領域を確保する */
    new = (Frac *)malloc(sizeof(Frac));
    /* s と b を比較して、
       確保した領域のメンバに引数の値を代入する */
    if(s < b) {
        new->bunshi = s;
        new->bunbo = b;
    } else {
        new->bunshi = b;
        new->bunbo = s;
    }
    /* 確保した領域の next を更新する */
    new->next = p->next;
    /* p の next は確保した領域を参照する */
    p->next = new;
}

int main(void)
{
    /* add_frac2() の動作確認 */
    Frac head2;
    head2.bunshi = 0; head2.bunbo = 0;
    head2.next = NULL;
    show(&head2, "head2 (1)");
    add_frac2(&head2, 4, 7);
    add_frac2(&head2, 5, 2);
    add_frac2(&head2, 9, 9);
    show(&head2, "head2 (2)");

    return 0;
}
```

■問 5

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    /* x 番目が最大文字長として残すようにする */
    int i, x;
    /* ひとまず 1 番目が最大文字長とする */
    x = 1;
    /* 1 番目以降の argv の文字列に対して繰り返す */
    for(i=1; i<argc; i++) {
        /* x 番目の文字列よりも i 番目の文字列が長い? */
        if(strlen(argv[x]) < strlen(argv[i])) {
            /* i 番目を最大文字長とする */
            x = i;
        }
    }
    /* 結果を出力する */
    printf("最も長い文字列: %s\n", argv[x]);

    return 0;
}
```