2018 年度 情報領域演習第三
— 第 6 回 —

注意事項

- 前回の課題で連結リストに関する復習のための演習を行ったが，回答状況を鑑み，今週も前回の課題を引き続き取り組んでもらう．課題を提出するコマンドは前回と同じであるが (前回の資料参照)，出席は 06を含む checkerコマンドであることに注意せよ．
- 次回までは第 5 回も第 6 回もどちらも提出できる状態であるため，間違って第 6 回のプログラムを第 5 回の checker コマンドで提出してしまって以前成功した第 5 回の提出物を上書きしないように注意すること．
- 連結リストは今後の演習においても非常に重要な概念である．問題文だけでなく，その前にある説明をよく読み，必要であれば参考図書等で自習するとよい．わからないことがあれば TA や教員に質問しよう．
- プログラムの形式が指定されている場合には，それに従うこと．従わなくても「成功」と表示されることがあるが，得点にはならない．また，採点の際は checkerとは異なる入出力例を用いて動作確認するので，「失敗」する反例に対する小手先の対策だけでは得点にならないことがあるので注意せよ．
- 演習時間終了時点での提出状況についても評価の対象となるので，時間内にできるだけ多くの問題に挑戦するとよい．

はじめに

前回同様，CED において以下のコマンド (青字の部分) を実行することで出席を表明できる．ただし，N は所属するクラス名 1，2，3 で置き換え，N の前には空白を入れないこと．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/06/checkerN
提出開始: 11 月 xx 日 14 時 40 分 0 秒
提出締切: 11 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2018-11-xx-14-58
 問題:    結果 | 提出日時          | ハッシュ値                       |
    1:  未提出 |                   |                                  |
    2:  未提出 |                   |                                  |
                        ⋮
```

出席を表明するには，授業の開始から 30 分以内に実行する必要がある．実行しても出席状況が「欠席」である場合は教員に伝えること．なお，上の出力は例であるので，実際の締切りは各自コマンドで確認せよ．

提出も同じコマンドを用いて以下のように実行する．ただし，N はクラス名 (1 から 3)，X は 1 から 5 のいずれかの問題番号であり，prog.c は提出する C プログラムのファイル名であり，X の前後には忘れずに空白を入れること．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/06/checkerN X prog.c
```

ここで，ファイル名として指定するのは，問題番号 X の問題を解く C プログラムのソースファイルであり，コンパイル済みの実行ファイルではない．ファイル名は特に指定しないが，「英数字からなる文字列.c」などとすることが望ましい．このコマンドを実行することにより，指定されたファイルがコンパイルされ，実行テストプログラムが自動的に起動される．コンパイルに失敗した場合にはエラーとなり，提出されたことにはならないので注意すること．コンパイルが成功した場合には複数回の実行テストが行われ，実行テストにも成功すると「成功」と表示される．実行テストに失敗すると「失敗」と表示されるとともに反例となる入力と出力が表示されるので，失敗した理由を見つけるための参考にするとよい．なお，入出

力が大きい場合やファイルとして入力したい場合には共に表示されるパスにあるファイルを使うと便利である．ただし，入力のない問題の場合は失敗しても反例は出力されない．

正しく提出できたか確認するためには以下のように出席の表明と同じコマンドを用いて確認できる．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/06/checkerN
提出開始: 11 月 xx 日 14 時 40 分 0 秒
提出締切: 11 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2018-11-xx-14-47
 問題:     結果 | 提出日時            | ハッシュ値                          |
    1:     成功 | 2018-11-xx-15-33    | 9b762c81932f3980cf03a768e044e65b |
                          ⋮
```

ハッシュ値は，提出した C プログラムのソースファイルの MD5 値である．CED では `md5sum` コマンドを用いて MD5 値を見ることにより，提出したファイルと手元のファイルが同じものであるかを確認することができる．

```
[p1610999@blue00 ~]> md5sum prog.c
9b762c81932f3980cf03a768e044e65b
```

なお，一度「成功」となった問題に対し，実行テストに失敗するプログラムを送信してしまうと，結果が「失敗」になってしまうので注意すること．

1 連結リストの応用

前回，連結リストに関する復習をしたが，今回もリストを使った応用問題にチャレンジしよう．連結リストは基本的なデータ構造であり，今後の演習においても自在に扱えることが要求される．

まず，連結リストの代わりに配列を使って，次の 2 つの問題を解いてみよう．

問題 1

入力で与えられる 2 つの文字列が表す正の整数に対し，足し算を行う問題である．標準入力は 2 行で与えられ，各行の文字列は最大 126 文字の数字 (つまり 126 桁以下の整数) であるものとする．入力桁数が大きすぎるため，入力された数値をそのまま C 言語の整数型 (`int` 型や `long int` 型や `long long int` 型) で読み込めない点に注意しよう．環境にも依存するが，最も大きな整数を表現できる `unsigned long long int` 型でも 2^{64} 未満に過ぎず，10 進数表せば最大でも 20 桁である．今回扱う入力は 126 桁 (1 無量大数 10^{68} よりはるかに大きい！) までの整数であるため，その各桁 (0 から 9 まで) を `int` 型の配列に格納して，2 つの整数の足し算を行う．入力される整数値は，1 の位を配列の 0 番めに，10 の位を 1 番めに，100 の位を 2 番めに，…というように 10^i の位を i 番めに格納する．たとえば，入力が `4321` と `98765` の足し算なら，一の位から順に `1`，`2`，`3`，`4` と格納された配列と，`5`，`6`，`7`，`8`，`9` と格納された配列を用意して，各桁の足し算を行い，その結果を標準出力に出力する (繰り上がりに注意せよ)．作成すべきプログラムの形式は問わないが，入力は 0 から 9 までの 1 桁ごとに配列に格納するものとし，必要であればサンプルプログラムを参考にするとよい．標準入力から入力される文字は数値と改行文字だけであり，2 行とも 126 個以下の数字が並び改行文字で終わるものと仮定してよい．

入力例

```
182
8585
```

出力例

```
8767
```

入力例

```
4321
98765
```

出力例

```
103086
```

入力例

```
123456789012345678901234567890
987654321098765432109876543210
```

出力例

```
11111111101111111111011111111100
```

ヒント

- 以下のプログラムは 1 行で与えられた整数値を桁ごとに配列に格納し，それを表示するプログラムである．

```c
#include<stdio.h>
#define DIGITS 126
char buf[DIGITS+2]; /* ← 改行文字とナル文字の 2文字分だけ多く buf を用意 */

int main() {
  int i, len;
  int arr[DIGITS] = {};          /* ←要素が全て 0 の配列を作るために {} と書く[1]  */
  i = 0;
  fgets(buf,sizeof(buf),stdin); /* ←改行までの文字列を buf に格納 */
  while(buf[i]!='\n') ++i;       /* ←buf の改行の位置を探す (入力の長さがわかる) */
  len = i;                       /* ←改行の位置を len に保存 */
  for(i=0; i<len; ++i)
    arr[i] = (int)(buf[len-1-i]-'0'); /* ←buf[len-1-i]を数値にしてarr[i]に格納 */

  printf("Your␣input␣is␣");
  for(i=len-1;i>=0;--i) printf("%d",arr[i]);  /* ←arr[i]の数値を上の位から表示 */
  printf(".\n");
  return 0;
}
```

- 1 つめの配列の各要素に対して，2 つめの配列の要素を足していって，繰り上がりをあとで処理するのが楽かもしれない．

問題 2

入力で与えられる 2 つの文字列が表す正の整数に対し，掛け算を行う問題である．問題 1 と同様に，標準入力は 2 行で与えられ，各行の文字列は最大 126 文字の数字 (つまり 126 桁以下の整数) であるものとする．右に示すような筆算の要領で掛け算を行うプログラムを書いてもよいし，

```
   123
 × 456
 ----
   738
  615
 492
 ----
 56088
```

$$\left(\sum_{i=0}^{N} a_i 10^i\right)\left(\sum_{j=0}^{N} b_j 10^j\right) = \sum_{k=0}^{2N}\left(\sum_{i=0}^{k} a_i b_{k-i}\right) 10^k$$

のような等式 (ただし，$i > N$ のとき $a_i = b_i = 0$) を利用して各桁を繰り上がりを無視して計算してから，最後に繰り上がりの処理を行ってもよい．

入力例

```
11111
11111
```

出力例

```
123454321
```

[1] C 言語で `int a[100] = {12,34};` と宣言すると，長さ 100 の配列のうち 0 番めが 12，1 番めが 34，2 番め以降が全て 0 である配列が作られる． つまり，宣言された数より少ない要素しか書かないと残りは 0 で埋められる． ただし，`int a[100];` だけでは各要素は初期化されない (何が入っているかわからない) ので注意．

入力例

```
123
456
```

出力例

```
56088
```

入力例

```
11111111111111111111 ← 1 が 20 個
88888888888888888888 ← 8 が 20 個
```

出力例

```
98765432098765432096790123456790123456 8
```

ヒント

- 入力を配列に格納する方法は問題 1 を参考にするとよい.
- 入力の 2 つの配列とは別に出力用の配列を用意する方がわかりやすい．入力は 10 進表現で最大 126 桁の数であるため，出力 (掛け算の結果) は最大 252 桁である．

さて，今度は同じ問題を配列ではなく連結リストを用いて解いてみよう．配列では 126 桁と限定していたが，連結リストではそのような制限が不要になるうえに，126 桁も使わない場合であれば少ない桁数の分だけの長さを持つリストを用意すれば済む．前回と同じく，各節点は以下の構造体を用いて与えるものとする．

```
1 struct node {
2   elementtype element;
3   struct node *next;
4 };
```

以下の問題では，いずれも要素の型は `int` 型である．

問題 3

問題 1 と同様に，入力で与えられる 2 つの文字列が表す正の整数に対し，足し算を行う問題である．ただし，入力の長さは限定されていないため，節点の要素として各桁の数を含む連結リストを用いて表すことにする．入出力例は問題 1 と同じであるが，実行テストでは桁数の大きな数値も入力となっている．ただし，入力の桁数が事前にわからないため，問題 1 で示した `fgets` 関数と `buf` を用いた方法はそのまま使えない [2] ので注意せよ．入力の処理には以下のプログラムを参考にするとよい．

```
1  #include <stdio.h>
2  int main() {
3    int d;
4    char c;
5    while((c=getchar())!='\n') { /* ←読み込んだ数値を c に入れて，改行文字まで以下を繰り返す */
6      d = (int) c - '0';         /* ← 読み込んだ数字を int 型として d に格納 */
7      /* ※ ここで読み込んだ数字 d を処理 (リストに挿入する， など) */
8    }
9    return 0;
10 }
```

問題 4

問題 2 と同様に，入力で与えられる 2 つの文字列が表す正の整数に対し，掛け算を行う問題である．ただし，入力の長さは限定されていないため，節点の要素として各桁の数を含む連結リストを用いて表すことにする．入出力例は問題 2 と同じであるが，実行テストでは桁数の大きな数値も入力となっている．

[2] もちろん，`char buf[1000];` などと宣言することで入力が 999 桁より小さいときはうまくいくかもしれないが，それでは配列の代わりに連結リストを用いる意味がなくなってしまう．

問題 5

今回は，C 言語の標準の整数型で表現できないような大きな整数 (多倍長整数) の加算と乗算に関する実装をテーマとした．問題 3 以降の実装を用いれば，(メモリが許す限り) どんな大きな整数の計算も可能であるが，リストの各要素の型 (`elementtype`) が `int` 型であるにもかかわらず，0 から 9 までしか使っていない (10 進 1 桁) 点で無駄がある．そこでもう少しメモリの利用効率を高めるために 10 進 4 桁を単位 (10000 進数) として計算を行う問題とする．たとえば入力の数 123456789 は 1 と 2345 と 6789 に分けて格納するため `node` の個数は (頭を除き)3 つですむ (問題 4 では 9 つの `node` を要する)．対象の `node` 数が減少 [3] することで計算時間も短縮できる．プログラムの実行時間は `time` コマンド [4] で計測できるので，長い入力例に対して時間差を体感するとよいだろう．なお，入出力に関しては問題 4 と同様とする．

メモ

実用的には，要素の型を `unsigned int` とし，各要素に 0 から $2^{32}-1$ までの値 [5] を格納するような 2^{32} 進表現を使えば，無駄のない整数の表現が可能になる．要素の型として `unsigned long long int` などを用いてもよい．ただし，計算途中の結果が溢れないようにすることと，型の上限値まで使う場合は「各桁を計算してから繰り上がりを後で処理する」ということができなくなるので注意が必要である．

[3] `cons()` の呼び出し回数を数えればよい．
[4] `time ./a.out` で実行時間が表示される．出力結果の表示が不要であれば `time ./a.out > /dev/null` とすればよい．
[5] `unsigned int` 型で表現できる整数の上限値は処理系によって異なる．