2006 年度プログラミング演習 3 概要

プログラミング演習３は必須科目です．単位を取得しないと卒業できませんので注意してください.

１．プログラミング演習３の目的

C 言語は, 手続き型言語の中では, 比較的早く開発されたにもかかわらず現在でも多くのプログラム開発で使われており，非常に長い生命力を持っている言語です．また，現在インターネット上で稼動するプログラムや GUI を使ったプログラミングで多く使われている C++や Java の制御文も，C 言語の制御文の形式がほぼそのまま採用されており，C 言語をマスターすればこれらのオブジェクト指向言語の修得も容易になります.

プログラミング演習３では，１回生のときに「データ構造とアルゴリズム」および「プログラミング演習２」で学んだ内容のうちの一部を，実際にプログラミングして，より複雑な機能を持ったプログラムを作れるレベルに到達することを目標にしています．データ構造とアルゴリズム」および「プログラミング演習２」で学んだ内容をしっかり復習しながら,「使える技術」として身に付けてください．プログラミングは，数学のように今までの基礎の上に次の技術を上乗せしていく「積み上げ技術」ですから，基礎ができていない人は，それ以上のことをやろうとしても無理です.

２．プログラム作成の手順

「プログラミング演習３」の目的はもうひとつあります．それは，プログラムを作成する手順を学ぶことです.

実際のプログラミングは，次のような手順で行います.

（１） ＜問題分解＞
与えられた問題を解くために，より小さくかつ解きやすい小問題に分解する．C 言語プログラムの場合，その小問題が関数を作り出すことであれば都合がいい.

（２） ＜データ構造＞
それぞれの関数を実装するために，都合の良いデータ構造を探したり，作り出したりする.

（３） ＜アルゴリズム＞
そのデータ構造のもとで, 問題解決のアルゴリズムを探したり, 考え出したりする.

（４） ＜単体プログラミング＞
関数として実装し，デバッグ用のメインプログラム（「テストドライバー」という.）を作り出してデバッグする.

（５） ＜統合プログラミング＞
すべての単体プログラム（関数）を使って，与えられた問題に対応したプログラムを作り，デバッグする.

（６） ＜整形＞
プログラムがわかりやすく，かつ無意味な部分がないように，整形する.

（７） ＜ドキュメンテーション＞
プログラムの外部仕様（使用方法），および内部仕様（構造）をわかりやすくまとめる.

現在，１回生の「データ構造とアルゴリズム」および「プログラミング演習２」を終えたばかりの皆さんは，C 言語の文法の基本と，いくつかの基本関数を組み合わせて，新しい関数を作り出す方法を学んだだけで,「プログラミングによって問題を解決する」能力は,

まだついていません．いままで出された問題は，実社会の問題を解決するというよりは，C言語を理解するために作られた練習問題だったわけです．「道具」の基本的性質を理解しただけで，道具を使って目的を達することを学んだわけではないのです．鉋（かんな）や鋸（のこぎり）の性質がわかっただけでは，家は建てられないように，文法を理解しただけではプログラミングで実社会の問題は解決できないのです．この「プログラミング演習３」では，「家（プログラム）の建て方」も学びます．

３．プログラミング演習３全体の課題

プログラミング演習３全体の課題は大きく２つあります．１つ目はC言語で「整数電卓」を作ること，2 つ目は C 言語で「クライアント／サーバ型のチャットシステム」を作ることです．これが立てるべき「家」だと思ってください．これらの課題は，実に重要なプログラミングの手順，データ構造，アルゴリズムの役割を，実践的に理解し実用的なシステムが開発できるようになるためのエキスが詰まっています．

４．プログラミング演習３の進め方

演習は次のルールに従って行ってください．

（１） プログラミングは各自独立して行います．

（２） Step 課題を順番に解決してください．余力があればオプション課題へと進んでください．

（３） 他人の書いたプログラムの全部または一部をコピーして提出したものは，**不正行為とみなし厳正に対処**します．文法書などからプログラムの一部を転記することはかまいませんが，その意味を理解しているかどうかは課題終了時にチェックされます．

（４） C 言語の文法書やオンラインマニュアルは自由に参照してください．

（５） 各課題ごとにプログラムが完成したら速やかに担当教員，またはその代理であるTA・ES に報告し，確認してもらってください．

（６） 演習レポートは，ソースプログラムとその解説，および実行結果をプリントして，締め切り日までに各自提出してください．**締め切り後の提出は大幅に減点**します．

５．プログラミング演習３に臨む心構え

演習は，次の心構えで臨んでください．

（１） プログラミング能力をつけることは，メディア情報学科のすべての学生の最低限の要求です．他人のプログラムをコピーしたり，他人に書いて貰ったりしたのではプログラミング能力は身につかないのは言うまでもありません．できないのにできる「振りをする」人が増えています．しかし，これは社会にでたらすぐばれてしまいます．仕事が分担されるからです．誰も自分の仕事に手一杯で，代わりにやってはくれません．本当の力をつけてください．

（２） 初期の段階では，プログラミング能力は，「掛けた時間」に比例します．プログラミングが苦手だと言う人も，時間をかけてみましょう．ほとんどの人ができるようになるはずです．

（３） 演習は，授業時間に演習室にくるだけでは駄目です．自分で講義時間以外にやってみることが重要です．また，友達と一緒に「ああでもない，こうでもない」と言ってやったら，楽しくかつ早くできるようになります．プログラム開発は競争ではありません．プロジェクトの中で助け合って，共通の目標を一緒に達成する仕事です．演習を通じて，いい友達を作ってください．

（４） プログラミングは，開発生産性と品質がとても重要です．プロの C 言語プログラマで，１ヶ月 1,500～2,000 行のプログラムを書きます．１ヶ月 1,000 行以下しか書けない人は，苦労することになります．同じものでも，早く作るにはどうしたら良いかに関心を持ちましょう．また，どうしたら「良い（わかりやすく，性能の良い）

プログラム」を書けるようになるかということに関心を持つことも重要です．早く良いプログラムが書ける人は，技術者としてそれだけ価値が高い人です．

６．プログラミング演習３の評価

評価の割合：

日常点（課題達成度点）40 点 ＋ レポート（4 回）40 点 ＋ 最終講義試験 20 点

単位取得の条件：

授業に 11.2 回以上出席すること．

遅刻については以下の基準により出席回数に換算します．

15 分以下は 1/3 回欠席，45 分以下は 2/3 回欠席，それ以上は 1 回欠席．

評価　90 以上　：　A＋

89-80　：　A

79-70　：　B

69-60　：　C

59 以下　または，出席回数が 11.2 回に満たないもの：F

レポートのコピーを見つけたら，上記の評価点にかかわらず厳格な処遇を行います．**課題のチェックは第１回から第 13 回目の演習時間中およびフォローアップティーチング中**にのみ行いますので，必ず決められた時間内にプログラムを完成させ課題を教員もしくは TA・ES に確認してもらってください．またオプション課題についても上記時間内に確認してもらった場合のみ加点対象としますので注意してください．

７．レポート課題と提出方法

課題と提出日時

第 1 回　課題：**字句解析**　　提出日時：5 回目の授業の前日（5 月 17 日）17:00 まで

第 2 回　課題：**2 項演算**　　提出日時：8 回目の授業の前日（6 月 7 日）17:00 まで

第 3 回　課題：**四則演算**　　提出日時：11 回目の授業の前日（6 月 28 日）17:00 まで

第 4 回　課題：**Server/Client**　提出日時：14 回目の授業の前日（7 月 19 日）17:00 まで

(注)　6 月 10 日（土）「木曜講義日」は予備日とする．

提出場所

クリエーションコア 1 階　レポート提出箱

８．レポートの書き方

レポートの記載内容

- 表紙 （氏名，学生証番号，クラス，課題番号，提出日）
- 課題レポート本体

　課題番号（Step?-?）

1. 機能 （何をするためのプログラムかを記述.）
2. 外部仕様 （第三者がプログラムを利用できるように，起動方法・入力方法・出力内容・制限事項について記述.）
3. 内部仕様 （変数と関数の説明．ただし，関数の引数の説明はプログラムソースリストにもコメントとして記述すること.）
4. 実行例 （制限事項などの仕様がわかる実行例を作成すること.）
5. 考察・感想
6. 参考文献
7. プログラムソースリスト

９．参考図書

平林 雅英 著　**ANSI C / C++辞典―ANSI / ISO / JIS 対応**
単行本（ソフトカバー）共立出版（1996.1）
価格：￥6,500

島内 剛一 編集　**アルゴリズム辞典**
単行本（ソフトカバー）共立出版（1994.8）
価格：￥25,000

平林 雅英 著　**新 ANSI C 言語辞典**
単行本（ソフトカバー）技術評論社（1997.5）
価格：￥2,300

M. J. Donahoo・K. L. Calvert 著　**TCP / IP ソケットプログラミング C 言語編**
単行本（ソフトカバー）オーム社（2003.5）
価格：¥1,800

2006 年度プログラミング演習 3
関数電卓課題
（第 1 週目から第 10 週目）

課題内容

ここでは，ＵＮＩＸのコマンドモードによる対話型で動作する電卓を作ります.

例として,

```
%calc
>12+5
17
>3++5
ERROR
>5/2
2
>2*(3+4)
14
>.
%
```

上の例で,

- 式を入力させるためのプロンプト（入力促進文字）として「>」を使います.
  （入力した式には，下線をつけて示してありますが，実際下線がつくわけではありません.）
- 間違えた式を入力すると,「ERROR」と出力されます．式の正しさは，通常の数学の規則に従います.
- 「.(ピリオド)」で終了します.

という規則を設けます.

この問題を解決するために 10 週間で下記の課題に取り組みます.

Step1: 字句解析　＜第 1 週目～第 4 週目＞
　　入力文字列を数値と演算子に分解する.

Step2: 2 項演算　＜第 5 週目～第 7 週目＞
　　Step1 にて解析した数値と演算子を使って実際に演算を行う.

Step3: 四則演算　＜第 8 週目目～第 10 週目＞
　　括弧付き演算を実現するためにスタックを用いて四則演算を行う.
　　（注）カッコ付きの数式を処理する場合には，優先順位に基づいて未処理のまま数値や演算子を保持する必要があります．例えば,「2*(3+4)」の場合,「2*(3」を読み込んだ段階で処理をせず，そのまま状態をスタックに保持しておき,「)」を読み込んだときに初めて演算処理を行います.

2006 年度プログラミング演習 3
関数電卓課題 1
～ 字句解析（文字列処理の復習） ～
＜第 1 週目～第 4 週目＞

はじめに

関数電卓課題 1 を達成するために，以下の 7 つの「ステップ課題」を設定します．なお，関数電卓課題 1 では Step1-7 が「レポート課題」となっています．

（Step1-1：「Echo プログラム」）

- 機能：
  画面で，入力した文字列と同じ文字列を出力する（エコーする）プログラム．1 文字目に「.(ピリオド)」を入力すると終了します．プロンプト（入力促進文字）として，「>」を使います．
- ユーザインタフェース：

```
%p1-1
>12+32
12+32
>4*5
4*5
>.
%
```

ヒント：

- キーボード入力からデータを受け取る方法は色々ありますが，ここでは，fgets を使って 1 回で受け取りましょう．fgets の仕様はプログラミング演習 3 概要に挙げてある参考書またはオンラインマニュアル(man fgets)を参照すること．
- この問題は，プロンプト → 式入力 → エコー → プロンプト →…と，無限ループで表現されることになります．C 言語で無限ループを作るのは，

```
#define EVER 1
  :
while(EVER){   }または，for(;EVER;){  }で書けます．これができない人は，
```

C 言語の文法書から見直しましょう．

（Step1-2：「文字と文字列」）

- 機能：
  入力文字列を文字に変換し 1 文字ずつ表示するプログラム
- ユーザインタフェース：(下線文字列は，ユーザがキーボードから入力したものを示す)

```
%p1-2
>12+34
1
2
+
3
4
\n (改行コード)
>5+8
```

```
5
+
8
\n (改行コード)
>.
%
```

ヒント：

- Step1-1 で作成したプログラムは入力文字を文字列として扱っていました．これを文字型変数として扱うにはどうすればよいか考えてください．
- これができない人は，C 言語の文法書から見直しましょう．

（Step1-3：「データ表示と管理」）

- 機能：
  入力文字列を１つずつ文字型変数（%c），整数型変数（%d）にて表示するプログラム．ただし改行コードは表示しなくて良い．
- ユーザインタフェース：

```
%p1-3
>12+34
%c=1, %d=49
%c=2, %d=50
%c=+, %d=43
%c=3, %d=51
%c=4, %d=52
>5+8
%c=5, %d=53
%c=+, %d=43
%c=8, %d=56
>.
%
```

ヒント：

- 内容自体は難しくありませんが，文字を文字型変数で扱う意味と整数型変数で扱う意味を良く理解してください．文字を%d で表示するということはアスキーコードを表示することと同じです．

（Step1-4：「値と演算子の分解」）

- 機能：
  入力文字列を数字と演算子に分解し，解析結果を返すプログラム．

0～9 の番号：	Numeric
四則演算子（ +, -, *, / ）：	Operator
空白：	Blank
改行コード：	無視（何もしない）
上記以外：	ERROR

  と表示してください．
- ユーザインタフェース：

```
%p1-4
>12+34
1 Numeric
```

```
2 Numeric
+ Operator
3 Numeric
4 Numeric
>12+*:  34
1 Numeric
2 Numeric
+ Operator
* Operator
: ERROR
  Blank
3 Numeric
4 Numeric
>5@891
5 Numeric
@ ERROR
8 Numeric
9 Numeric
1 Numeric
>.
%
```

ヒント：

- 文字はすべてアスキーコードにより管理されています．よってアスキーコードによる比較を行えばこの問題に対処できるはずです．アスキーコード表では数字の 0～9 は順番に並んでいるので対象文字が変数 num に格納されているとすると，if('0' <= num && num <= '9') を使って比較すれば変数 num がアスキーコード上で数字かどうか判定できます．

（Step1-5：「桁数の計算」）

- 機能：
  入力文字列を数字と演算子およびエラーに分解し，数字の桁数を計算するプログラム．
  0～9 の数字：　　Numeric
  四則演算子（ +, -, *, / ）：Operator
  空白：　　Blank
  改行コード：　　無視（何もしない）
  上記以外：　　ERROR
  と表示し，Numeric の場合のみ桁数を数える．
- ユーザインタフェース：

```
%p1-5
>12+34
1 Numeric  1
2 Numeric  2
+ Operator
3 Numeric  1
4 Numeric  2
>12+*:  34
1 Numeric  1
2 Numeric  2
+ Operator
* Operator
```

```
: ERROR
  Blank
3 Numeric  1
4 Numeric  2
>5@891
5 Numeric  1
@ Error
8 Numeric  1
9 Numeric  2
1 Numeric  3
>.
%
```

ヒント

- Step1-4 で作成したプログラムの結果に対して連続する数値を数えるだけです．ここでも改行コードについては表示する必要はありませんが，最後に改行コードが入っていることを良く理解しておいてください．

（Step1-6：「字句解析」）

- 機能：

入力文字列を数値と演算子に分解し，結果を表示するプログラム．数値および演算子はすべて「文字列」としてデータ管理すること．また以降空白については無視すること．

数値：	Numeric
四則演算子（ +, -, *, / ）：	Operator
空白：	無視（何もしない）
改行コード：	無視（何もしない）
上記以外：	ERROR

と表示する．

- ユーザインタフェース：

```
%p1-6
>12+34
12 Numeric
+ Operator
34 Numeric
>12+*: 34
12 Numeric
+ Operator
* Operator
: ERROR
34 Numeric
>5@891
5 Numeric
@ ERROR
891 Numeric
>.
%
```

ヒント：

- Step1-5 で作成した数字の桁数を数えるプログラムを改良し，数字（1 文字）を数値（文字列）として管理する問題．**入力配列とは別にもう１つ配列を用意して数値の桁数分および演算子，エラーを切り出せば解決できる．**ただし文字列としてデータ管理するので最後に「\0」が必要になることに注意すること．これを忘れるとセグメンテーションエ

ラーとなる．図１に切り出し文字列のイメージを図示する．この切り出し処理を繰り返し行うことでこの問題に簡潔に対処できる．

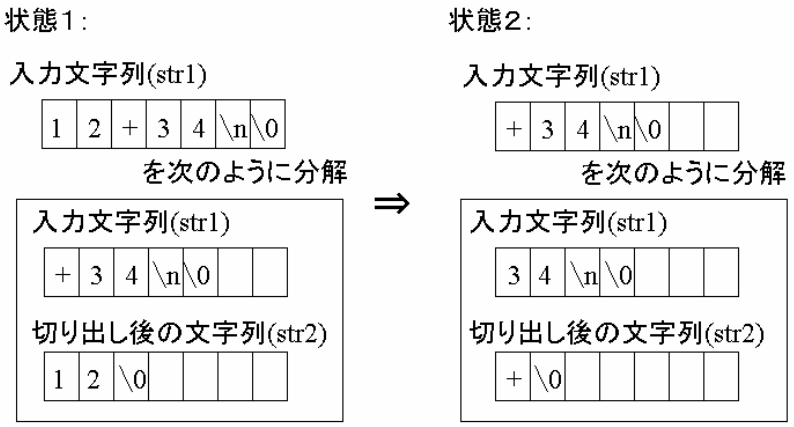


図１：入力文字列の切り出しイメージ

（Step1-7：「ソースプログラムの整形」）

- Step1-6 で作成したソースプログラムを，次の規則に従って，整形し，読みやすいものにしてください．整形の規則は，

（１） 変数名，関数名：
英単語で統一し，空白の代わりに大文字を使って語をつなぐ方法（ハンガリアン命名規則）で，それが何を意味しているかを他の人にとっても分かる名前をつけて示す．また，データ型を示す識別子を名前の先頭につけます．（例，`intVariableCounter`, `strToken`）．1～2 文字の意味のわからない変数名や関数名は使わないでください．

（２） 空行，空白：
プログラムのブロック間に空行を入れて，ブロックの塊が一目で分かるようにします．また，入れ子（ループ）構造のレベルをインデントで示します．さらに演算オペレータに前後に，空白を入れて，字句レベルの塊が一目でわかるようにします．

（３） コメント行，行末コメント：
ブロックが何をしているかはコメント行で示し，行が何をしているかは行末コメントで示します．変数には，それが何であるかの行末コメントが必ず必要です．また，ブロックには，その機能や役割をコメント行（複数行でもいい）で示します．

（４） ブロック化：
プログラムは，それが何をしているかが分かりやすいようにするためで，関数化はこの１つの方法です．だらだらと長いプログラムを書かないこと．関数化は，適切に行われないと，引数がやたら多くなったり，プログラム処理の流れ（ロジック）がかえって分かりにくくなるので注意しましょう．

（５） ヘッダー：
プログラムの先頭行に，プログラム名，機能概要，作成年月日，作成者，版，更新履歴，使用法を書きます．

この他，Ｃ言語特有のテクニックとして，

（６） `#define` 文を使って，マジックナンバー（意味のわからない定数）がプログラム中

に出現するのを防ぐ方法があります．ここの問題では式の文字列が入る領域を定義しておく必要がありますが，

```
char formula[100];
```

と書いたのと，

```
#define MaxLength 100
char formula[MaxLength];
```

と書いたのと比べてみてください．後者の方がずっとわかりやすいので，必ず後者の方法をとってください．

（7） `#include <stdio.h> /*stdin, printf */`
のようにライブラリ定義の後に，使っている関数を書いてください．これでどんな関数を使っているかがわかります．

（8） ループの範囲を示すため，ループの終わりにどんなループかをコメントで書いてください．

```
while(EVER){
  :
} /* While */
```

などがあります．

これらの方法を使って，Step1-6 で作成したプログラムを整形してください．どうですか？プログラムが格段に読みやすくなったでしょう．今後は，プログラムが完成してから後に整形するのでなく，整形する必要のないようにはじめから書いていきましょう．

以上

2006 年度プログラミング演習 3
関数電卓課題 2
～ 2 項演算 ～
＜第 5 週目～第 7 週目＞

はじめに

関数電卓課題 2 を達成するために，以下の 4 つの「ステップ課題」を設定します．なお，関数電卓課題 2 では Step2-4 が「レポート課題」となっています．

（Step2-1：「字句解析の関数化」）

- 機能：
  入力文字列を数値と演算子に分解し，解析結果を返す関数．数値は正の整数だけとします．ここでは表示する数値や演算子はこれまでと同様に文字列のままで問題ありません．
  数値： Numeric
  四則演算子（ +, -, *, / ）：Operator
  空白，改行コード： 無視（何もしない）
  上記以外： ERROR
  と表示すること．
- ユーザインタフェース：

```
%p2-1
>12+34
12 Numeric
+ Operator
34 Numeric
>12+*:  34
12 Numeric
+ Operator
* Operator
: Error
34 Numeric
>45
45 Numeric
>45^20
45 Numeric
^ ERROR
20 Numeric
>.
%
```

ヒント：

- この問題の基本的な処理の流れは，図 2 のフローチャートのようになります．
- トークンの判定は，多岐（6 分岐）になるので，if 文よりは，switch 文を推奨します．
- 項の種類を判定する関数（judgeToken 関数）を作成し，それを getToken 関数内で呼び出して判定してください．また judgeToken 関数の戻り値である判定結果は下記のように予めプログラムの先頭で定義しておけば簡潔にプログラムを作成できます．

```
#define Numeric     1
#define Operator    2
#define Blank       3
        …
```

- 最初は，判定の無いプログラムを作って，デバッグし，それが正常に動くようになったら，判定部分を付け加えて，完成させます．大きなプログラムは，このように少ずつ確実なものを拡張していくことが重要です．
- `getToken` 内の引数の `str1` は，参照だけがされる文字列領域ですが，`str2` は，そこに値（トークン）が書き込まれる領域です．このような引数を持つ関数は，あらかじめその引数領域を呼び出す側で宣言しておかなければなりません．これは，ポインタとしての宣言ではだめで，文字型を持った領域（文字列）でなければなりません．
- 文字列は，終端位置に，「`\0`」が入っていなければならないことに注意しましょう．これを忘れると，「セグメントエラー」になります．

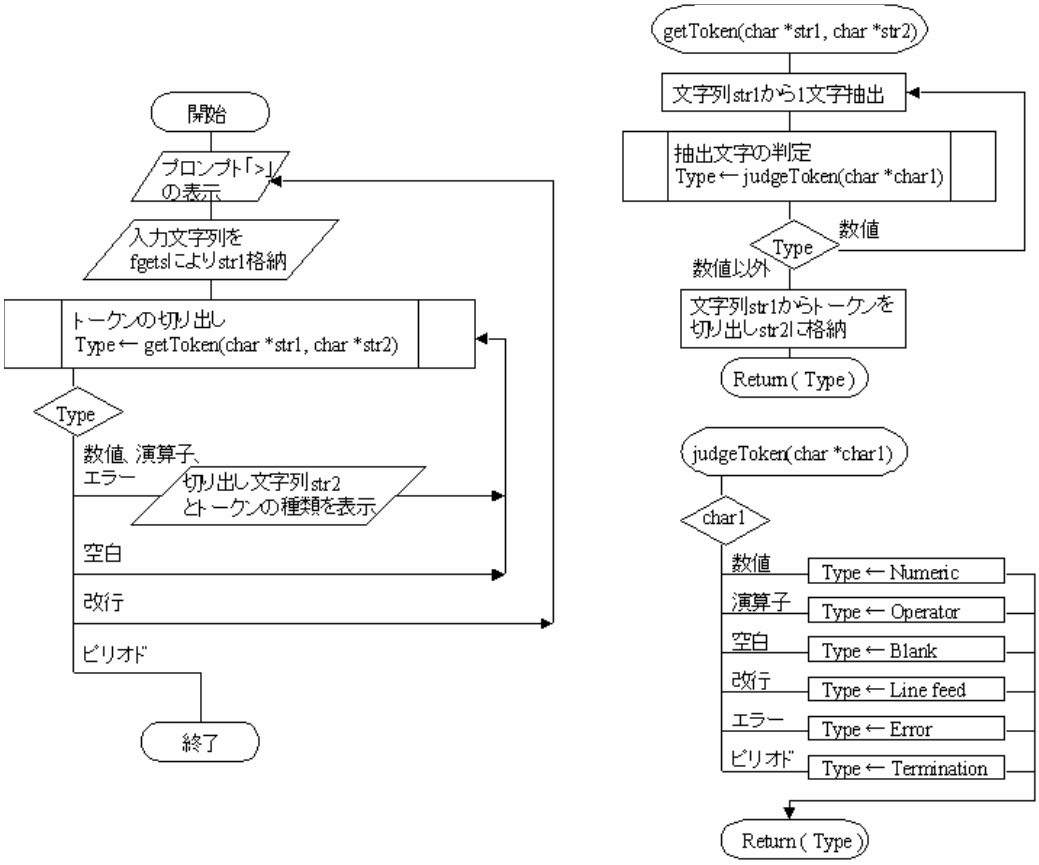


図２： Step2-1 のフローチャート

（Step2-2：「トークンの型変換」）

- 機能：
  Step2-1 の解析結果（トークン）に対して，数値は整数型変数（`int` 変数），演算子は文字型変数（`char` 変数）に変換するプログラム．値は正の整数だけとします．ここでは数値は`%d`（整数型），演算子とエラーは`%c`（文字型）にて表示してください．

  数値<%d>： `Numeric`
  四則演算子（ +, -, *, / ）<%c>： `Operator`
  空白，改行コード： 無視（何もしない）
  上記以外<%c>： `ERROR`
- ユーザインタフェース：
  `%p2-2`

```
>12+34
12 Numeric
+ Operator
34 Numeric
>12+*:  34
12 Numeric
+ Operator
* Operator
: Error
34 Numeric
>45
45 Numeric
>45^20
45 Numeric
^ ERROR
20 Numeric
>.
%
```

ヒント：

- 文字列を整数型変数にキャストするには atoi( )を使えば簡単に行える．詳細は教科書かオンラインマニュアルを参照のこと．

（Step2-3：「2 項演算」）

- 機能：
  式を解析した後，指定された演算を行い，計算結果を返すプログラム．
  空白と改行コードについては無視すること．また，四則計算(+, -, *, /)以外の演算子や，数値，空白と改行コード以外の文字が入力されると「ERROR」を出力します．
- ユーザインタフェース：

```
%p2-3
>12+34
46
>4*5
20
>45^20
ERROR
>.
%
```

ヒント：

- 数値と演算子の解析に基づいて演算を行うだけです．また式中に 1 つでも ERROR 文字があれば，その計算を行う必要はありません．（演算を行う前にエラーチェックを行えば簡単に課題を行えます．）また，カッコ演算や 3 項以上の多項演算などはここでは考慮する必要はありません．

（Step2-4：「ソースプログラムの整形」）

- Step2-3 で作成したソースプログラムを，整形の規則に従って，整形し，読みやすいものにしてください．整形の規則は，Step1-7 と同じです．

以上

2006 年度プログラミング演習
関数電卓課題 3
～　四則演算　～
＜第 8 週目～第 10 週目＞

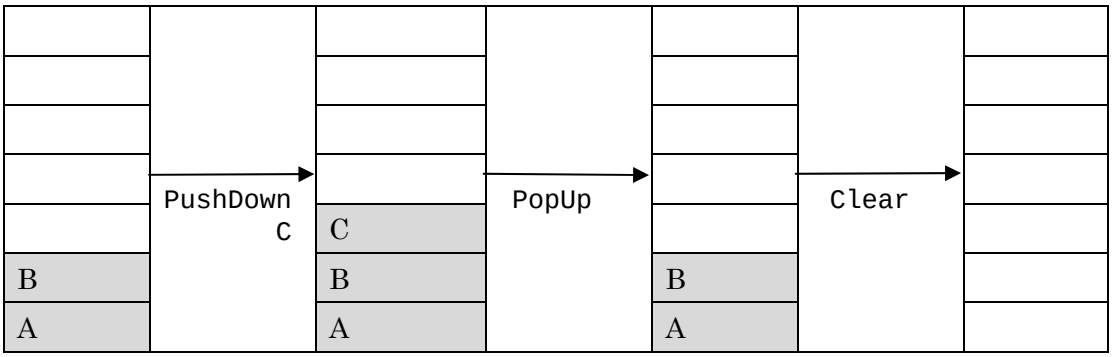
カッコ付きの数式や加減乗除の混合演算を処理する場合には，優先順位に基づいて，未処理のままトークンを保持する必要があります．例えば，「45*(23+10)」の場合，「45*(23」を読み込んだ段階で処理をすることはせず，そのままトークンの内容をスタック保持しておき，「(」を読み込んだときにはじめて演算処理を行います．

スタックとは

スタックは，「First-In Last-Out」の処理を可能とするデータ構造の一種で，机の上に積まれた書類のように最初に置かれたものが最後に取り出されます．スタックを実現するために，次の 3 つの基本操作が必要になります．

- Clear ：　スタックを空にする．
- PushDown：　データを記憶させる．
  ただし，記憶させたいデータはコマンドの引数として指定する必要があります．
- PopUp：　最も新しく記憶したデータを読み出す．

スタックの動作を理解するには，次のようなダイアグラムをイメージしてください．



はじめに

関数電卓課題 3 を達成するために，以下の 5 つの「ステップ課題」を設定します．なお，関数電卓課題 3 では Step3-5 が「レポート課題」となっています．

（Step3-1：「PushDown 関数」）

- 機能：
  スタックを配列で定義し，PushDown 関数と，スタックメモリをクリアする関数 clear の作成．（PopUp 関数は Step3-2 で作ります．）スタックはグローバル変数として定義してください．終了コマンドは End とします．
  なお，PushDown， Clear および End コマンド以外の文字列に対しては，エラーを表

示するようにします．下記ユーザインタフェースの下線部が入力文字となります．

- ユーザインタフェース：

```
%p3-1
>Clear
Depth=0
Stack={}
>PushDown
Parameter? 32
Depth=1
Stack={32}
>PushDown
Parameter? +
Depth=2
Stack={+,32}
>End
%
```

ヒント：

- コマンドの照合には strcmp( )を使いましょう．
- PushDown 以外のコマンドは引数がありませんので，Pushdown が呼び出されたときのみ PushDown するパラメータを要求します．
- また PushDown 関数は，スタックオーバフローなどのエラーが発生する可能性があるため，int 型の関数で，戻り値として 0 以外の値（例えば-1）が返ってくればエラーであるものとします．
- スタックの内容を表示する関数は，次の Step3-2 でも使いますし，それ以降もデバッグ用として便利なので，独立した関数 printStack( )にしておきましょう．

（Step3-2：スタック動作確認プログラム）

- 機能：
  Step3-1 のプログラムに，PopUp 関数および PopUp コマンドの処理機能を追加する．PopUp 関数は，正常動作の場合に 0 を，スタックに何も記憶されていないときにエラーを表す「0 以外の値（例えば-1)」を，戻り値として戻すものとします．
- ユーザインタフェース：

```
%p3-2
>Clear
Depth=0
Stack={}
>PushDown
Parameter? 32
Depth=1
Stack={32}
>PushDown
Parameter? +
Depth=2
Stack={+,32}
>Poup
ERROR
>PopUp
Output +
Depth=1
Stack={32}
```

```
>End
%
```

（Step3-3：「数式字句解析」）

- 機能：
  式を値と演算子に分解し，解析結果を返すプログラム．四則計算とカッコ以外の演算子や，整数以外の値を使うと「ERROR」と出力します．
- ユーザインタフェース：

```
%p3-3
>12*(4+51)
12 Numeric
* Operator
( Left Parenthesis
4 Numeric
+ Operator
51 Numeric
) Right Parenthesis
>45^20
45 Numeric
^ ERROR
20 Numeric
>.
%
```

ヒント：

- Step2-2 で作成した getToken 関数にカッコを考慮すれば完成です．「開始括弧」は Left Parenthesis,「終了括弧」は Right Parenthesis と表示しましょう．

（Step3-4：「電卓プログラム」）

- 機能：
  入力された式を解析した後，指定された演算を行い，計算結果を返すプログラム．
  四則計算とカッコ以外の演算子や，整数以外の値を使うと「ERROR」と出力します．
  また Step3-4 では多重カッコなどは考慮する必要はありません
- ユーザインタフェース：

```
%p3-4
>12*(4+51)
660
>4*5
20
>45^20
ERROR
>.
%
```

ヒント：

- 括弧つきの数式を実行するのに，スタックを使うことにより，種々の文型の処理が飛躍的に簡単になります．
  例えば，上の例で，

```
12*(4+51)
```

  を実行するとき，

入力

トークン	処理	スタックの中身
12	PushDown	{12}
*	PushDown	{*,12}
(	PushDown	{(,*,12}
4	PushDown	{4,(,*,12}
+	PushDown	{+,4,(,*,12}
51	PushDown	{51,+,4,(,*,12}
)	PopUp, PopUp, PopUp, PopUp, PushDown 55 （注１参照） （2 回目の PopUp で得られる演算子の種類に基づき, 3 回目の PopUp で得られる数値と 1 回目の PopUp で得られる数値を計算し, 結果を PushDown します.）	{55,*,12}
\n	（スタックに入っているトークンの数によって処理が異なります．注 2 参照） PopUp, PopUp, PopUp （2 回目の PopUp で得られる演算子の種類に基づき, 3 回目の PopUp で得られる数値と 1 回目の PopUp で得られる数値を計算）	{}

となります.

注１： ここでは括弧内の演算を 2 項演算限定としたので,「)」が入力されたときにその前の 3 つのトークンを見て処理を行うことができましたが，通常は 3 項以上からなる演算の処理が必要です．その場合には加減算より乗除算が優先されるとか，べき乗は乗除算より優先されるなどの処理が必要なので，これほど簡単ではありません.

注２： 「改行」が入力された時点での，受理できる文と文型およびスタック内のトークン数の関係は以下の通りです.

受理できる文	文型	スタック内のトークンの数
終了文	.(ピリオド)	1
演算文	数値 演算子 数値	3

（Step3-5：「ソースプログラムの整形」）

- Step3-4 で作成したソースプログラムを，整形の規則に従って，整形し，読みやすいものにしてください．整形の規則は，Step1-7 と同じです.

(注) すべて完成した学生は Web 上のオプション課題に取り組んでください.

URL http://www.media.ritsumei.ac.jp/students/index.html

オプション課題 1：実数を考慮した関数電卓

オプション課題 2：2 進数による関数電卓

オプション課題 3：log, sin, cos などの関数演算の追加

オプション課題 4：メモリー機能（代入処理）の追加

オプション課題 5：多項演算に拡張

以上