情報応用特論 I

講義ノート

－　２００１年度版　－

福岡工業大学
情報工学部 情報工学科
柴田望洋

BohYoh Shibata
Fukuoka Institute of Technology

本資料について

◆　本資料は、2001 年度・福岡工業大学大学院修士課程情報工学専攻の講義

『情報応用特論Ⅰ』

の補助テキストとして、福岡工業大学 情報工学部 情報工学科 柴田望洋が編んだものである。

◆　参考文献・引用文献等は、資料の最後にまとめて示す。

◆　諸君が本資料をファイルに綴じやすいように、研究室の学生達（卒研生と大学院生）が時間を割いて、わざわざ穴を開けるという作業を行っている（一度のパンチで開けることのできる枚数は限られており、**気の遠くなるような**時間がかかっている）。
必ずＢ５のバインダーを用意して、きちんと綴じていただきたい。

本資料は、Microsoft 社のワープロソフトウエアである Microsoft Word 2000 を用いて作成した。

第１回

平成 12 年 4 月 12 日

配布資料：

■ 『プログラミング言語Ⅱ』より pp.1〜5

■ 柴田望洋ら『明解Ｃ言語入門編 例解演習』より pp.94〜101

本講義は、いわゆる《講義形式》の授業です（レポートを課すかもしれませんが、そのときは簡単なものにしたいと考えています)。

まずＣ言語の問題（一部）を解いてもらいます。学部１年生〜２年生程度の問題です。

特にＴＡをしている学生であれば、立場上解けなければならない程度の基本的な問題です（少なくとも応用問題とはいえません)。しかし、諸君の多くは

『思ったほどは解けなかった』

と、多少ショックを受けたことでしょう。

意地悪を言えば、みなさんは就職活動の面接で『Ｃ言語は使えますね。』と聞かれると、たぶん『はい。』と元気に答えるはずですね。その直後に、この問題を解くように指示されたら、どうしますか？

《プログラミング言語》あるいは《プログラミング》を理解すること・学習すること・指導することについて、私は次のように考えています。

> いったん習得してしまえば簡単な、足し算や分数の計算などは、何度も何度も多くの問題を解いてきましたね。英語を学習するときは、文の中の単語を一つだけ入れかえることによって別の文を作ったり、同じ意味を表す別の言い回しによって文を作ったりして理解を深めましたね。
>
> プログラミング言語の学習でも、特に基礎の段階においては、数多くの問題に触れることが必要である、と私は考えています。したがって、一般的なテキストに示されている数少ないプログラムだけを頼りにして一通りの学習をすることは困難でしょう。
>
> 柴田望洋ら『明解Ｃ言語入門編 例解演習』, ソフトバンクパブリッシング, 1999

さて、みなさんは《プログラミング言語》あるいは《プログラミング》について、どのように接してきましたか？　現在、どのくらいのレベルに達していますか？

ちなみに、私の人生における目的の一つは、《プログラミング言語》を始め、情報関連のいくつかの科目について、世界最高（最良）の教材を作ることです。

気を取り直して、学部２年生向けのプリント『プログラミング言語Ⅱ』p.2〜p.5のプログラムを打ち込んで実行しましょう。これは、数当てゲームプログラムです。第１版〜第４版まであります。少しずつプログラムが複雑になるとともに、ゲームとして楽しいものとなります。

ただし、これらはC++言語ですから、すべてＣ言語に移植しながら打ち込んでください。ちなみに、第４版は以下のようになります。

```
/*
    数当てゲーム（その４）
*/

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

int main(void)
{
    const int max = 10;       /* 入力制限回数 */
    int  x;                   /* 読み込んだ値 */
    int  no;                  /* この数を当てさせる  */
    int  cnt = max;           /* 残り何回入力できるか？ */
    time_t   t;

    srand(time(&t) % RAND_MAX));     /* 乱数の種を初期化 */
    no = rand() % 100;               /* 0〜999の乱数を発生 */

    do {
        printf("残り%d回です。¥n", cnt);
        printf("整数を入力せよ：");
        scanf("%d", &x);
        cnt--;

        if (x > no)
            printf("¥a大きいです。¥n");
        else if (x < no)
            printf("¥a小さいです。¥n");
    } while (x != no && cnt > 0);

    if (x == no) {
        printf("正解です。¥n");
        printf("%d回で当たりましたね。¥n", max - cnt);
    }

    return (0);
}
```

乱数を発生させるのが **rand** 関数です。

最初に解いてもらった問題がのっている『明解C言語入門編 例解演習』や、乱数の発生法については、柴田望洋後援会オフィシャルホームページ

http://www.BohYoh.com/

の『著書』『C言語講座』『C／C++ FAQ』などのページを見てください。

さて、以下に示すのは、三つの整数の最大値を求める関数です。不等号をひっくり返すと最小値を求めることができます。

```
int max(int a, int b, int c)
{
     int m;

    m = a;
    if (b > m) m = b;
    if (c > m) m = c;
    return m;
}
```

それでは、三つの整数の中央値を求める関数を作ってください。　…　結構難しいですね。少なくとも日本語レベルでは、最大値・最小値・中央値はほとんど同レベルの問題に感じますが。一見簡単そうな問題も、実は難しいかもしれません。

この問題は、来週までの宿題といたします。

第2回

平成13年4月20日

配布資料：26枚

- ■ 『情報応用特論Ⅰ 講義ノート』　表紙／pp.0〜4　(5枚)
- ■ 『情報基礎ゼミナール』　表紙／pp.0〜4　(5枚)
- ■ 『文字の入力法』　表紙／pp.0〜4　(5枚)
- ■ 『ソフトウェア工学』　表紙／pp.0〜3　(4枚)
- ■ 『プログラミング言語Ⅱ』　表紙／pp.6〜11　(7枚)

『情報基礎ゼミナール』

弁証法・認識論の大家である南郷継正の論文（著書）からの引用を示しています。みなさんも、《大志を抱き論理能力を磨け》！

『文字の入力法』

学部1年生のTAをやっている人なんかは感じると思いますが、初心者はキーボードを打つのが大変です。したがって、このような資料が必要なのですね。

ところで、p.3は指使いを示しています。皆さん知っていますね？　はい、おそらく知っているようですが、それでは正しい指使いでキーボード打っていますか？　あらっ？今度は反応が悪いですね。

私はパソコンを初めて間もない頃、我流の間違った指使いを行っていました。どんどんスピードアップしましたが、途中でひっかかるのですね。それからタイピング練習の本を買ってきて、矯正し猛特訓しました。

私が（ノリノリのときは)、頭で文書を考えながら、あるいはプログラムを作りながら、目にも止まらぬ速度でキーボードを打ちます。しかも《ここでif文を使って…》なんて考えるより前に手が動きます。知人に、《ただ滅茶苦茶に指を動かしているかと思ったら、本当にちゃんと打ってるんですね。》とびっくりされたことがありますが、そうでしょうか？

音楽家のインプロビゼーション＝即興では、頭で考えるより先に音楽を奏でたりしますよね。あれと同じです。

『ソフトウェア工学』

本年度から、久しぶりにこの講義を担当しています。単なる“プログラミング”ではなく、ドキュメントも含めたソフトウェアの開発技術、そのためのプロジェクトの進め方などが対象です。

質と**量**、**製品**と**人間**、**概念**と**具体的手順**といった三つの対となる概念を含みます。

先週の課題について

さて、先週の宿題である三値の中央値を求めるプログラムは作ってきましたか？　最大値・最小値とは異なり、数多くの実現法のバリエーションが考えられます。

ここでは、二つの実現法を示します（画面にて／このプリントでは三つ）。

```
/*
    三値の中央値を求める
*/

#include  <ctime>
#include  <iostream>

using namespace    std;

typedef int(*mfnc)(int, int, int);

//--- Version 1 ---//
int med1(int a, int b, int c)
{
    if (a > b)
        return (a <= c ? a : b > c ? b : c);
    else
        return (b <= c ? b : a > c ? a : c);
}

//--- Version 2---//
int med2(int a, int b, int c)
{
    if ((b >= a && c <= a) || (b <= a && c >= a))
        return (a);
    else if ((a > b && c < b) || (a < b && c > b))
        return (b);
    return (c);
}

//--- Version 3 ---//
int med3(int a, int b, int c)
{
    if (a > b)
        if(c > a)
               return (a);
        else if (b > c)
               return (b);
        else
            return (c);
    else if (c > b)
        return (b);
```

```
    else if (a > c)
        return (a);
    else
        return (c);
}
//--- 実行 ---//
void go(mfnc f, int no, int loop)
{
    int  a[] = {1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3 };
    int  b[] = {2, 2, 3, 1, 1, 2, 2, 2, 3, 3, 1, 2, 2 };
    int  c[] = {2, 3, 2, 2, 3, 1, 2, 3, 1, 2, 2, 1, 2 };

    cout << "Versioin." << no + 1 << endl;
    for (int i = 0; i < 13; i++)
        cout << f(a[i], b[i], c[i]);
    cout << endl;

    clock_t start = clock();
    while (loop-- > 0)
        for (int i = 0; i < 13; i++)
            f(a[i], b[i], c[i]);

    clock_t end = clock();
    cout << "計算時間＝" << (double)(end - start) / CLOCKS_PER_SEC << endl << endl;
}

int main(void)
{
    const int  LOOP = 10000000;
    mfnc med[] = {med1, med2, med3};

    for (int i = 0; i < 3; i++)
        go(med[i], i, LOOP);

    return (0);
}
```

ちなみに implementation は**実現**あるいは**実装**という日本語があてられます。

このプログラムは、1998 年に制定された ISO/IEC 規格の C++言語規格の仕様にのっとって作られたものです。C++特有の事項や、関数へのポインタなどを理解していないと、このプログラムは分からないかもしれませんが、中央値を求める関数の部分は、C 言語と同じです。

main 関数では、a, b, c について 13 パターンの値を用意しています。大小関係としては、これが全てであり、全てのパターンに対して、中央値として 2 を返却すれば、その関数が正しく中央値を求めると（ほぼ）みなすことができます。

さて、Version 2 は、あまり効率がよくなく、実行に時間がかかります。何故でしょうか？最初の if 文の判定

```
if ((b >= a && c<= a) || (b <= a && c >= a)
```

に着目しましょう。ここで **b >=a** および **b <= a** の判定を、裏返した判定（実質的に同一の判定）が、続く else 以降で

```
else if ((a > b && c < b) || (a < b && c > b)
```

と行われます。つまり、最初の if 文が成立しなかった場合、同じ判定を再び繰り返すのですね（さて、等値や論理演算の回数は皆さんで比較してください。宿題です)。

この例から、いろいろなことが分かりますね。《三値の中央値を求める》という、一見簡単な問題も、いろいろな実現法があること、実現法によって効率が異なること等。

ソフトウェアを開発する過程は、現実の世界で与えられた要求あるいは問題を、仕様に変換し、それを具体的なプログラムへと投射する作業です。最終的に作成されるソフトウェアの品質は、その作業に依存するのです。

あるパソコンでの計算時間を比較した表を示します。

	Visual C++6.0	**C++ Builder 5.0**
Version 1	**6.519**	**7.311**
Version 2	**8.302**	**9.093**
Version 3	**7.03**	**8.272**

※いずれも、実行速度を向上させるコンパイルオプションを指定してコンパイルしたものです。

それでは、各実現法における、演算回数を比較してみてください。これは宿題としましょう。

Version 1

a	1	1	1	2	2	2	2	2	2	2	3	3	3
b	2	2	3	1	1	2	2	2	3	3	1	2	2
c	2	3	2	2	3	1	2	3	1	2	2	1	2
等値													
論理													
代入													
計													

Version 2

a	1	1	1	2	2	2	2	2	2	2	3	3	3
b	2	2	3	1	1	2	2	2	3	3	1	2	2
c	2	3	2	2	3	1	2	3	1	2	2	1	2
等値													
論理													
代入													
計													

Version 3

a	1	1	1	2	2	2	2	2	2	2	3	3	3
b	2	2	3	1	1	2	2	2	3	3	1	2	2
c	2	3	2	2	3	1	2	3	1	2	2	1	2
等値													
論理													
代入													
計													

第3回

平成13年4月26日

配布資料：30枚

- ■ 『情報応用特論Ⅰ 講義ノート』 pp.5〜9 (5枚)
- ■ 『情報基礎ゼミナール』 pp.5〜6 (2枚)
- ■ 『インターネット入門』 表紙／pp.1〜3 (4枚)
- ■ 『文字の入力法』 pp.5〜8 (4枚)
- ■ 『ソフトウェア工学』 pp.4〜6 (3枚)
- ■ 『プログラミング言語Ⅱ』 pp.12〜14 (3枚)
- ■ 『アルゴリズムとデータ構造』 表紙／pp.0〜7 (9枚)

『情報応用特論Ⅰ　講義ノート』

さて、三値の中央値を求める手順における演算回数はカウントしてきましたか？　誰もやってないですね。私もやっていませんから、あまり非難できないかもしれません。

私が大学生のとき、遅刻に対して非常に厳しい先生がいらっしゃいました。学生が遅刻して入室しようものなら「出て行けぇ〜。」と怒鳴られます。誰も遅刻しません（遅れたら怖くて入室できませんから)。しかし、一回だけですが、その先生自身が遅刻したことがあり、15分くらい遅れて謝りながら入室されました（私は心の中で、先生に対して「出て行けぇ〜。」と怒鳴りました)。

さてさて、この先生にとって遅刻の《基準》とは何でしょうか？　正確に時刻に対して遅刻かどうかを判定するのであれば、教員は必ずチャイムより前に教室に入っていなければなりません。しかし、多くの教員はそうではなく、《自分が入室した時刻》を基準に判定を行っているように感じられます。これは、おかしいですね。

『情報基礎ゼミナール』

（詳細省略）

『インターネット入門』

（詳細省略）

『文字の入力法』

記号文字の読み方、フォントの等幅やプロポーショナルという言葉などは、ちゃんと知っておきましょう。また、半角カタカナは御法度です。

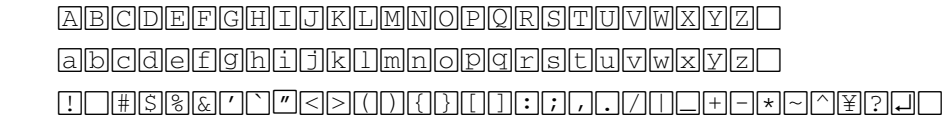
『ソフトウェア工学』

プリントの通りです。仕様・正確さなどの概念は、非常に重要です。

『アルゴリズムとデータ構造』

このプリントで、アルゴリズムやデータ構造について学習していきます。ちなみに、改行を表すこの文字 ␤ は、この週末に数時間かけて作ってきました。ちなみに、MS-Windows などにも含まれている Symbol というフォントには、↵という記号文字があります。この記号文字に対して、ワープロや DTP ソフトで、枠囲みをかければ、同等なものを作ることができます。しかし、ソフトによっては、そのような機能はありませんし、枠囲みの高さや幅に融通がきかないソフトもあります。

このような文字を全部作っておけば、あとあと便利です。ソフトによらず利用できます。ある意味では、どこにでも使いまわしの効く、とても再利用性の高い部品ともいえます。



さて、階乗値や最大公約数を、再帰呼出しを用いずに求めるのは、課題とします。是非解いてみてましょう。

『プログラミング言語Ⅱ』

本日お配りしたプリントには、分布をグラフ（記号文字*を並べる）として表示するプログラムを示しています。横向きグラフに比べて、縦向きグラフは少し難しくなります。このくらいであれば、すぐにプログラムが組めるようになっておかなければなりません。

第4回

平成13年5月10日

配布資料：36枚

■	『情報応用特論Ⅰ』	pp.10〜11	(2枚)
■	『アルゴリズムとデータ構造』	pp.8〜13	(6枚)
■	『Word入門』	pp.0〜6	(7枚)
■	『情報基礎ゼミナール』	pp.7〜8	(2枚)
■	『インターネット入門』	pp.4〜12	(9枚)
■	『ソフトウェア工学』	pp.9〜13	(5枚)
■	『プログラミング言語Ⅱ』	pp.15〜19	(5枚)

『Word入門』

『入情報基礎ゼミナール』

UNIXとC言語との関係は、関係者が一番詳しく知っています。

『インターネット入門』

Yahooは、『ヤフー』と読みます。『ヤッホー』と読まないようにしましょう。ちなみに、

oo は、ウかウーと発音するのであって、オーと発音しないように。

と教える英語の先生が多いようです。

確かに、前者は、book, good, took など、後者は boom, cool, pool, wool, school, scoop, spoon, too, tool などの語句がそのように発音します。単語が

しかし、世の中には例外が存在します。brooch はブロウチです（ちなみに brood はブルード)。

『ソフトウェア工学』

（詳細省略）

『プログラミング言語Ⅱ』

エスケープシーケンス（拡張表記）について、全部知っておきましょう。

『アルゴリズムとデータ構造』

今回も再帰。次回も再帰のお話です。

第5回

平成13年5月17日

配布資料：31枚

■	『情報応用特論Ⅰ 講義ノート』	p.12	(1枚)
■	『アルゴリズムとデータ構造』	pp.14〜23	(10枚)
■	『Word入門』	pp.7〜10	(4枚)
■	『文字の入力法』	pp.10〜14	(5枚)
■	『プログラミング言語Ⅱ』	pp.20〜24	(5枚)
■	『ソフトウェア工学』	pp.14〜19	(6枚)

『Word入門』

少なくともTAをやっている学生であれば、この資料に書かれていることくらいは、全部知っておかなければなりません。

さて、日本語ワープロは、数年前まで一太郎が主流でした。私のワープロ変遷を紹介しながら、日本語ワープロの歴史の一部を簡単に紹介しましょう。

まず、私が使った最初のワープロは、【自作ワープロ】でした。最初に買った（親の脛(すね)を齧りました）パソコンがPC-8001mkIIです。ちなみに、そのパソコンは漢字を表示すると、瞬時に出るのではなく、描画している様子がダラッと見えるんですね。NECのPCは、テキスト画面とグラフィック画面が別々でなのですが、PC-8001mkIIでは、グラフィック画面にのみ漢字を描画できます。したがって、BASICのプログラムで

```
100 PRINT "漢字"
```

と命令することすら、不可能なのです。

パソコン始めてから約３ヶ月でBASIC言語によってワープロを作りました。当時は仮名漢字変換ソフトなんかありませんでしたので、カナ漢字変換部も自分で作成しました。

JISコードでは、3020〜3024までが《ア》亜、唖、娃、阿、3025〜28が《アイ》哀、愛、挨、始 … と読みの順で並んでいます。そこで、各読みの先頭の読みと、コードをBASICのデータ文として用意します。

```
100 DATA "ｱ", "ｱｲ", "ｱｳ", …（以下省略）
200 DATA 3020, 3025, 3029, ……（以下省略）
```

このデータを参照して、候補を表示して選択させることによって、仮名漢字変換をするのです。もちろん、JISコードでの読みにしか対応していませんので、“動かす”は、いったん《ドウ》で《動》の文字を選択し、それから《か》《す》と入力します。

さて、パソコンはPC-9801F2、PC-9801VM2…と買い換えていきました。PC-98で最初に購入したワープロが【文筆Ver.II】です。当時は58,000円で、上付き文字、下付き文字が利用できる画期的なワープロでした。ただし、動詞などは終止形でないと変換できません。

すなわち《動かす》は、いったん《動く》でへんかんした後、《く》を消して《かす》と入力します。ちなみに、その頃比較的流行っていた、【松】は 128,000 円でした（もちろん、上付き文字、下付き文字は使えない)。

その後、【テラⅢ世】、そのバージョンアップ版である【Queen】を使いました。これらは、いずれも文書ファイルが 32Kバイトまでという制限がありました。後で知ったのですが、【文筆】を作ったプログラマが、別の会社から出したのが【テラⅢ世】、【Queen】だったのですね。どんどん機能は増えていきましたが、やっぱり似ていました。その作者は高校生くらいから、プロとして活躍していた人だったそうです。

さて、私が【Queen】を使っている頃は、既に【一太郎】が主流となっていました。【Queen】の機能や操作性に限界を感じた私は、【P1EXE】（ピーワン・エグゼ）を使いました。このワープロ、一太郎と互換性があり（一太郎の文書ファイルが読める)、グラフィクスなどの表現力など、一太郎に対して遥かに多機能でした。しばらくこのソフトを使っていましたが、バージョンアップして【P1EXE Plus】になった途端に、動作が重くなり、結局【一太郎 Ver.4】を使うことに決定しました。

そこで起こったのがバグ騒動。朝おきて NHK の 6 時のニュースを見ると、「ジャストシステム社が発売する一太郎というワープロソフトウェアに大量の**不具合**があり、問題となっています。」　…　ここで問題となったのが、ジャストシステムの姿勢。Ver.3 発売後に随分と日数が経過しており、【P1EXE】などのライバルが台頭してきたため、なんと大量のバグがあることを分かっていながら出荷したのです。

私も経験しましたが、ある操作をした後に、単語登録をすると暴走するとか、保存したファイルが、オープンできないとか…。

一太郎は、約半年をかけて【Ver.4.1】、【Ver.4.2】とバージョンアップをし、やっと【Ver.4.3】で安定しました。

このことから、いろいろな教訓を学ばなければなりません。

・あるバージョンが広く長く使われていると、ユーザの期待が高まります。開発者は、その期待に対するプレッシャーを感じます。

・そこで大幅な改良・機能追加、あるいは一からの作り直しなどが行われます。そうすると、前バージョンと、データ（文書ファイル）の互換性が損なわれたり、操作性の変更などが余儀なくされます。ユーザは戸惑います。

《**バージョンアップ**》は、その中身だけでなくスケジュールなども含めて、複雑な要因が絡む、大変な作業なのです。

さて、私は【Ver.5】、【Ver.6】、【Ver.6.3】と一太郎を使いつづけました。【Ver.6.3】の時代も随分と長かったです。発売延期を重ねた、待望の【Ver.7】が発売された日は、昼休みに大学近くのパソコンショップに購入に行きました（今はつぶれてカラオケショップになっていますね)。開けてびっくり、使ってガックリ。

【Ver.6.3】で作成した文書ファイルを開くと、罫線などの配置がずれる、メニューなどの操作性が大幅に変更されている。さらに全体的に重い。さらに、特に罫線の前後など、Delete キーや Back Space キーで文字を消すと、それに反応するのに、場合によっては 3 秒くらいかかる！！のです。もちろん、普通に瞬時に文字が消えるときもあります。

操作に対する応答までの時間＝反応速度にムラがあるのは、ユーザの直感を逆なでするものであり、非常に使いにくい！

プログラムの内部的な品質が低いことは、見て取れました（はっきり言えば、下手糞なプログラム)。

その時、私は「あれだけの時間と労力を費やして、この程度のソフトウェアしか開発できないのであれば、この会社は潰れるな。」と感じました。

操作性が異なるのも、スピードが遅いのも、以前のバージョンの文書ファイルを読み込んだ場合レイアウトが崩れやすいのも《**バク**》でなく《**仕様**》なのでしょうか？世間一般的に騒がれることはありませんでした（ちなみに【Ver.7】は修正版が出された)。

まっ、とにかく【一太郎 Ver.7】を購入した、その日に、私は【Word】に鞍替えすることを決意しました。

…もっとも【Word】に非がないわけではありませんが、現在まで使いつづけています。さらに、文書によっては、Page Maker, Frame Maker, InDesign といったソフトウェアを利用しています（いろいろなソフトを使っていると、ときどき操作を間違えてしまいます)。

『ソフトウェア工学』

プリントの通りです。

『文字の入力法』

（詳細省略）

『プログラミング言語Ⅱ』

プリントの通りです。

『アルゴリズムとデータ構造』

プリントには書いていませんが、『原問題』、『問題を分割する』といった用語を覚えておきましょう。

第6回

平成13年5月24日

配布資料：36枚

■	『情報応用特論Ⅰ 講義ノート』	pp.13〜15	(3枚)
■	『アルゴリズムとデータ構造 第3章』	pp. 1〜12	(12枚)
■	『情報基礎ゼミナール』	p.9	(1枚)
■	『Word入門』	pp.11〜13	(3枚)
■	『文字の入力法』	p.9	(1枚)
■	『インターネット入門』	pp.14〜16	(3枚)
■	『プログラミング言語Ⅱ』	pp.25〜31	(7枚)
■	『ソフトウェア工学』	pp.20〜21	(2枚)
■	『C++によるオブジェクト指向』	pp.0 〜3	(4枚)

『Word入門』

（詳細省略）

『情報基礎ゼミナール』

（詳細省略）

『インターネット入門』

（詳細省略）

『ソフトウェア工学』

（詳細省略）

『C++によるオブジェクト指向』

ジャンケンプログラムを改良していきましょう。

『文字の入力法』

（詳細省略）

『プログラミング言語Ⅱ』

ある意味で、関数はプログラムの“部品”です。

（詳細省略）

『アルゴリズムとデータ構造』

本日は、基本ソート（単純交換／単純選択／単純挿入）のお話です。

まず、プログラム中の

```
#define swap(type, x, y) do {type t; t=x; x=y; y=t; } while (0)
```
○

を考えましょう。これは、《type 型の x と y の値を交換する》関数形式マクロです。このマクロは、各種のテキストなどでは、よく以下のように定義されています。

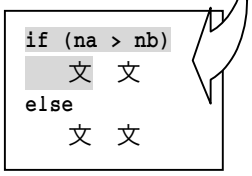
```
#define swap(type, x, y) {type t; t=x; x=y; y=t; }
```
×

しかし、これはお薦めできません。下の図を考えましょう。

```
if (na > nb)
    swap(int, a, b);
else
    swap(int, a, c);
```

→ マクロ展開

```
if (na > nb)
    { type t = na;  na = nb;  nb = t; } ;
else
    { type t = nb;  nb = na;  na = t; } ;
```



if 文は、以下のいずれかの形式をもちます。

if（式）文

if（式）文 else 文

したがって、最初の文の後に else があれば、後者の形式とみなされ、そうでなければ前者の形式とみなされ、そこで if 文は終了となります。さて、この場合、na > nb が成立したときに実行されるのは、{から}までの複合文であって、網掛け部が前者の if 文とみなされます。この直後に else が位置すべきですが、セミコロンが蛇足となってしまいます。というにも、セミコロンだけの文＝空文とみなされるからです。

したがって、どうしてもこのマクロを使いたいのであれば、右のように、セミコロンを抜かなければなりません。… でも、これは、どうみてもＣ言語のプログラムらしくありませんし、ついついセミコロンを付けたときはエラーとなります。

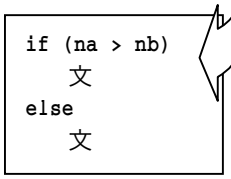
```
if (na > nb)
    swap(int, a, b)
else
    swap(int, a, c)
```

さて、プリントの定義を用いると、右のように展開されます。while に続く( )中の式の値は 0 ですから、{ }で囲まれた部分が実行されるのは、1 回限りです。また、do 文は、

do 文 while (式);

という形式をもっていますので、構文的に右のように解釈され、全く問題がないのです。

```
if (na > nb)
    do { type t = na;  na = nb;  nb = t; } while (0);
else
    do { type t = nb;  nb = na;  na = t; } while (0);
```



さて、プログラム右の部分に着目しましょう。

```
int x[7];
int nx = sizeof(x) / sizeof(x[0]);
```

nx の初期化子に使われている **sizeof(x)** は配列 **x** の全体の大きさで、**sizeof(x[0])** は、要素 **x[0]** の大きさです。

したがって、**nx** は、**x** の要素数 **7** で初期化されることになります。したがって、プログラム中、これ以降は、配列の要素数が必要な箇所には、**7** でなく **nx** と書かれています。このようにしておけば、配列の要素数を変更する際は、**x** の宣言だけを変更すればよいわけです。

それでは、右のように宣言したら、どうなるでしょうか。これでも、**nx** はきちんと **7** で初期化され、問題ありません。

```
int x[7];
int nx = sizeof(x) / sizeof(int);
```

しかし、《配列の要素に格納する値が大きくなった。**int** 型でなく、**long** 型に変更しよう》と仕様の変更が行われた際に、右のように **x** の宣言だけを書きかえたら、おかしなことになってしまいます（もちろん、処理系によっては、**sizeof(int)** と **sizeof(long)** が等しいこともあり得ますので、たまたまうまくいくかもしれませんが）。

```
long x[7];
int nx = sizeof(x) / sizeof(int);
```

したがって、要素数を格納する変数が必要な場合は、このプログラムのように宣言するテクニックを使いましょう。

第7回

平成13年5月31日

配布資料：25枚

- ■ 『情報応用特論Ⅰ 講義ノート』　pp.16〜18　(3枚)
- ■ 『アルゴリズムとデータ構造 第3章』　pp.13〜21　(9枚)
- ■ 『プログラミング言語Ⅱ』　pp.32〜37　(6枚)
- ■ 『ソフトウェア工学』　pp.22〜24　(3枚)
- ■ 『C++によるオブジェクト指向』　pp.4 〜7　(4枚)

『プログラミング言語Ⅱ』(第4回の補足)

p.18のプログラムをコンパイル・実行してみましょう。文字列がテロップのように流れます。拡張表記(エスケープシーケンス)'¥r'は、復帰を意味し、その行の先頭にカーソルを位置付けます。

p.24 の問題(4-1)の解答を示します。setwという処理子を挿入することによって、続く出力の桁数を指定することができます。

```
#include <iomanip.h>
#include <iostream.h>

int main(void)
{
    for (int i = 1; i <= 9; i++)
        cout << setw(i) << i << '¥n';

    return (0);
}
```

```
1
 2
  3
   (中略)
        9
```

それでは、これをC言語で実現してみてください。…私は、あっという間に2バージョン作成しました。右に示すのが2重ループを用いた解答です。適当な数だけのスペースを表示して、数値を表示します。

```
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i <= 9; i++) {
        for (j = 1; j <= i; j++)
            putchar(' ');
        printf("%d¥n", i);
    }
    return (0);
}
```

もう一つの実現例が下に示すものです。**printf** 関数を

```
printf("%5d", 123);
```

と呼び出したら、□□**123** と表示しますね（□はスペース)。

```
#include <stdio.h>

int main(void)
{
    int i;

    for (i = 1; i <= 9; i++)
        printf("%*d¥n", i, i);

    return (0);
}
```

5 桁といった桁数の指定は定数でなく、引数として変数的に指定することができるのです。それが*です。

したがって、この **printf** 関数の呼び出しは、『**i** の値を最低 **i** 桁で表示してください。』と依頼していることになります。

第8回

平成13年6月7日

配布資料：34枚

■ 『情報応用特論Ⅰ 講義ノート』	p.19	(1枚)
■ 『アルゴリズムとデータ構造 第1章』	pp.1〜11	(11枚)
■ 『プログラミング言語Ⅱ』	pp.38〜42	(5枚)
■ 『情報基礎ゼミナール』	p.10〜11	(2枚)
■ 『Word入門』	pp.14〜15	(2枚)
■ 『インターネット入門』	p.13,pp.17〜19	(4枚)
■ 『ソフトウェア工学』	pp.25〜29	(5枚)
■ 『C++によるオブジェクト指向』	pp.8 〜11	(4枚)

--- 昨年度の『講義ノート』より抜粋 ----------

『文字の入力法』

□ Windows上でのIMEの起動／終了は、Alt+半角／全角で行います。“Alt”は、alternateの略ですね。“交代する”、“互い違いになる”などの意味を持ちます。たとえば、普通にFキーを押すと、文字fが入力されますが、Altキーを押しながらFキーを押すと、Windowsの大部分のソフトでは、ファイルメニューが開きます。Fキーに与えられた、別の機能を働かせるためにalternateするわけです。

□ 先週確認しましたが、諸君の一部は、キーボードタイピングの正しい指使いは、頭では知っていても、なかなか実践していませんでしたね。日本語には、《身に付ける》という言葉があります。別に体にピッタリくっつくわけではありません。その技術に習熟し、身体レベル・意識レベルで完全に『自分のものになる』ということですね。

この他にも、「あの人は《腹黒い》。」、「腹が立つ」、「《腸(はらわた)が煮え繰り返る》思いをした」、「身にしみる」など、身体を意識する言語が数多くあり、これは外国語には、ほとんどみられない表現です。このような言語を使って、日本人は身体に対する意識を高め、文化を構築してきました（最近、このような言語が使われなくなり、文化も急速に失われつつあるのが現状ですが)。

□ 私は、ジャストシステム社のATOKの方が、MS-IMEより良いと感じますが、MS-IMEを使っていますし、学部の学生にも、こちらを教えています。理由は「MS-IMEは、Windowsを買うとただで付いてくるから。」です。

『データ構造とアルゴリズム』

★『アルゴリズム＋データ構造』 はプログラムではない ★

1994 年にＣマガジン誌に発表した拙文から引用しよう[1]。

“Algorithms ＋ Data Structure ＝ Programs〃‥‥これは、Pascal の設計者として知られる Dr. Niklaus Wirth が著した、あまりにも著名な本のタイトルである（邦訳：片山卓也訳『アルゴリズム＋データ構造＝プログラム』，日本コンピュータ協会，1979)。プログラムとは、アルゴリズムとデータ構造を組み合わせたものであるという考え方は、最近の新しいプログラミングパラダイムとは必ずしも一致するわけではないし、そもそも発想が要素主義的であるという大きな欠点があるものの、ある意味では、プログラミングの本質を鋭く捉えたものである。

要素主義という言葉について理解せねばなるまい[2]。

要素主義・実証主義

現象のより小さく細分化された要素について、その性質を規定する条件を単純且つ少数のものに絞り、そこから極めて明白性の高い「個々の事実」の即自的、帰納的法則化を目指す立場。

関係主義・全体主義

現象をより大きな一纏まりの全体として見做し、要素に先立つ〝構造〟或いは要素間の〝連関〟、要素の〝変化〟、更にはそれらの総合たる〝構造変動〟等々の法則を捉えようとする立場。

■ 参考文献 ■

1) 柴田望洋,『アルゴリズムとデータ構造Ｉ』,
Ｃマガジン，Vol.6，No.4，pp.28-58，1994
2) 高岡英夫,『武道の科学化と格闘技の本質』，恵雅堂，1987

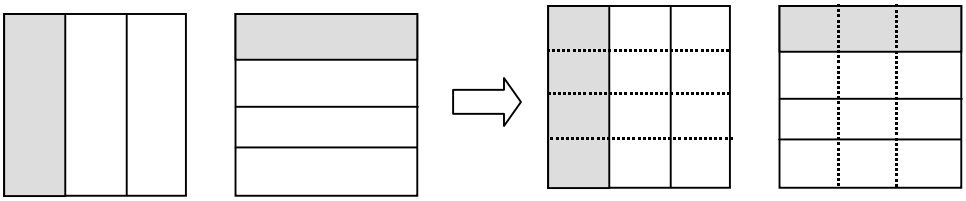
私の拙文から、知って欲しいこと。

◆　学問の流れは、【要素主義】から【関係主義（全体主義)】へと確実に移っています。

◆　他人を貶(けな)すときは、最初に批判して、最後に持ち上げましょう。
『彼は言ったこと無視するし○○だし△△だよ。でも、彼はいい人だよね。』
と最後の一言が大事です。これが、
『彼はいい人だよね。でも、彼は言ったこと無視するし○○だし△△だよ。』
では駄目です。

本日は、表紙と前書きだけです。昨年の資料を大幅に改良してから配りたいと思っていますので、本文は間に合いませんでした。
プログラミング言語も、アルゴリズムも、どんな教科でも、生徒が自分自身で《発見する》ように学習できるような教材・教育法でなければなりません。

小学生に 1/3 + 1/4 という分数の足し算を教えるとするとしましょう。みなさんは、どのように《通分》を教えますか。
『分母の値が異なるときは、そのまま足せないから、分母どうしを掛け合わせて…』
と手順だけを説明しても、なかなか理解してもらえないし、いったん理解しても、すぐに忘れてしまうかもしれません。
四角のケーキがあります。欲張りなＴ君は、Ｔ君含めて 3 人のグループＡでケーキを等分して貰い、Ｔ君含めて 4 人のグループＢでもケーキを等分してもらいました。Ｔ君がもらったケーキの合計の大きさは？
下のような図を使って説明すればよいですね（説明の仕方は自分で考えましょう)。



第９回

平成 13 年 6 月 14 日

配布資料：46 枚

■	『情報応用特論Ⅰ 講義ノート』	p.20〜22	(3 枚)
■	『アルゴリズムとデータ構造 第 1 章』	pp.12〜21	(10 枚)
■	『プログラミング言語Ⅱ』	pp.43〜48	(6 枚)
■	『Excel 入門』	pp.0〜16	(17 枚)
■	『ソフトウェア工学』	pp.30〜31	(2 枚)
■	『C++によるオブジェクト指向』	pp.12〜19	(8 枚)

--- 昨年度の『講義ノート』より抜粋 ----------

『情報応用特論Ⅰ』

増分演算子（インクリメント演算子）には、前置と後置の２種類があります。前置

```
++a
```

では、この式全体の値の評価が行われる前に、インクリメントが行われ、後置

```
a++
```

では、この式全体の値の評価が行われた後に、インクリメントが行われます。

したがって、a の値が 3 であるとき、

```
b = ++a;
```

を実行すると、まず a がインクリメントされ、値が 4 となります。式++a を評価した値は、4 ですから、b には 4 が代入されます。

また、同様に a の値が 3 であるとき、

```
b = a++;
```

を実行すると、式 a ++を評価した値である 3 が b 代入されます。その後、a がインクリメントされ、値が 4 となります。

```
(3-13)
#include <stdio.h>

int main(void)
{
    int i, no;

    printf("何個表示しますか：");
    scanf("%d", &no);

    i = 1;
    while (++i <= no)
        switch (i % 3) {
          case 1 : putchar('A'); break;
          case 2 : putchar('B'); break;
          default: putchar('C'); break;
        }
    return (0);
}
```

『インターネット入門』

・インターネットは、本来《ネットワークのネットワーク》という意味です。そもそも、inter は、international などの単語にも使われます。その inter とはどういう意味か、来週まで調べておきましょう。

『インターネット入門』

16 ビット漢字コード JIS コード、シフト JIS コード、EUC コードの違いは知っていますか？ JIS コードは、他の半角文字と区別が付かないから、文字列に埋め込むときに、KANJI-IN や KANJI-OUT などの制御文字が必要となりますので、文字列の見かけ上の長さと、物理的な記憶域上の長さに食い違いが生じます。そこで、漢字の先頭 8 ビットを未使用領域に入るよう、コードをずらしたのがシフト JIS コードですね。

漢字変換プログラムくらいは、20〜30 分くらいもあれば作れるようになっておかないといけません。機会があれば、この講義でもやりましょうか。

『プログラミング言語Ⅱ』

「x の値が 0 であれば○と表示し、そうでなければ×と表示」

三つの実現例が示されています。この中で C や C++に特有なのが、《実現例１》です。

C と C++の if 文

```
if (式) 文1 else 文2
```

では、()中の式の値が非 0 であれば文 1 を、そうでなければ文 2 を実行します（他の言語では、判定の式は、真偽を表す論理型でなければならないことが多いようです)。

したがって、《実現例１》が最も素直といえますし、熟練した C/C++プログラマは、このような表記を好みます。

等しいかどうか、等しくないかどうかを判断する演算子である==と!=は、成立すれば 1、そうでなければ 0 という値を生成します。したがって、《実現例２》は、もし x が 0 であれば、式 x == 0 を評価した値が 1 となり、式の値が非 0 であるから○と表示される、という２段階を踏むことになります（ただし、多くのコンパイラは、実現例１と同等なコードを出力しますので、実行速度が低下するといった心配は、まず不要です)。

《実現例１》

```
if (x)
    cout << "○¥n";
else
    cout << "×¥n";
```

《実現例２》

```
if (x == 0)
    cout << "○¥n";
else
    cout << "×¥n";
```

《実現例３》

```
if (x != 0)
      cout << "×¥n";
else
    cout << "○¥n";
```

『文字の入力法』

さて、皆さん栗田先生の《指回し》。これ、お奨めです。これをやるだけで（特に小中学生などの若い人は)、視力がアップしたり、柔軟性が高められたりと、いろいろな効果があります。やり始めの頃はイライラしますが、なれると、首や肩の血流が増加し、心が落ち着きますし、頭も良くなり、ボケ予防にもなります。皆さんも是非やってください。

ちなみに、私は二十代後半くらいから、いろいろな能力開発法、健康法などを実践してきました。昔仮性近視だった視力も、現在は 2.0 です。鍛えれば、人間の能力はドンドン伸びていきます。

『プログラミング言語Ⅱ』

(3-40) 右に示すようなやり取りによって、要素数が5である配列の各要素に、先頭から順に整数値を読み込み、50以上80未満である要素を列挙するプログラムを作成せよ。

このプログラムの実現例を示します。左側は辻君、右側が私のものです。

```
#include <stdio.h>

int main(void)
{
    int x[5], i, num = 0;

    printf("５個の整数を入力せよ。¥n");
    for (i = 0; i < 5; i++) {
        printf("No.%d : ", i + 1);
        scanf("%d", &x[i]);
        if (x[i] >= 50 && x[i] < 80)
            num++;
    }
    printf("50以上80未満は%d個です。¥n", num);
    for (i = 0; i < 5; i++) {
        if (x[i] >= 50 && x[i] < 80)
            printf("No.%d = %d¥n", i + 1, x[i]);
    }
    return (0);
}
```

```
#include <stdio.h>

int main(void)
{
    int x[5], z[5], i, num = 0;

    printf("５個の整数を入力せよ。¥n");
    for (i = 0; i < 5; i++) {
        printf("No.%d : ", i + 1);
        scanf("%d", &x[i]);
        if (x[i] >= 50 && x[i] < 80)
            z[num++] = i;
    }
    printf("50以上80未満は%d個です。¥n", num);
    for (i = 0; i < num; i++) {
        printf("No.%d = %d¥n", x[i]+1, z[x[i]]);
    return (0);
}
```

左側のプログラムは、整数を読み込みながら、50以上80未満の要素数をカウントします。その後、もう一度配列の全ての要素に対して、50以上80未満であるかどうかを調べながら表示します。

右側のプログラムは、整数を読み込みながら、50以上80未満の要素の添字を配列zの先頭から順に格納していきます。最後の表示では、配列zの内容をもとに表示を行います。

左側のプログラムの欠点は、配列全体を2度走査することです。10,000個のデータで、50以上80未満であるという条件を満たす要素が数個程度しかない場合、２度の10,000回もの繰返しは無駄です。また、全ての要素に対して、2度条件判定を行うことも無駄のようです。また、これは配列の走査だから良いですが、実際にディスクなどに記憶されているデータの判定だったら、ディスクをガチャガチャ読み込む作業にかかる時間などは少なくありません。

一方、右側のプログラムは、条件判定が行われるのは、最初のfor文だけであり、また、2度目のfor文は、条件を満たした要素の個数だけ繰り返されという点で左側のものよりも優れています。ただし、《配列が余分に必要である》ため、より多くの記憶域を必要とするものとなっています。

第 10 回

平成 13 年 6 月 21 日

配布資料：28 枚

- ■ 『情報応用特論Ⅰ 講義ノート』 pp.23〜25 (3 枚)
- ■ 『アルゴリズムとデータ構造 第 1 章』 pp.22〜25 (4 枚)
- ■ 『プログラミング言語Ⅱ』 pp.49〜51 (3 枚)
- ■ 『情報基礎ゼミナール』 pp.12〜17 (6 枚)
- ■ 『ソフトウェア工学』 pp.32〜35 (4 枚)
- ■ 『C++によるオブジェクト指向』 pp.20〜27 (8 枚)

--- 昨年度の『講義ノート』より抜粋 ----------

『C++プログラミング』

このプログラムは、C++の開発者である B.Stroustrup 氏の著書から引用した電卓プログラムです。『コンパイラ』や『アルゴリズムとデータ構造』を学習したのであれば、このくらいのプログラムは作れないといけません。電卓は、非常に簡単なコンパイラともいえます。

ちなみに、コンパイラは、どのようなことを行うのでしょうか？

```
if (a > b) c = a; else c =b;
```

このような文字の並びを、トークン（単語）に区切っていきます。これが**字句解析**の段階です。この段階で綴り間違いや、キーワードや定数などの単語とみなせない語句があればエラーとなります。

```
if ( a > b ) c = a ; else c = b ;
```

単語に分割した後は、**構文解析**を行います。C 言語の if 文は、

```
if (式) 文
if (式) 文 else 文
```

のいずれかの形式ですから、これにのっとっているかどうか、と《形》として正しいかどうかが調べられます。この場合は OK ですね。

それが終了すると、次は**意味解析**です。たとえば、C 言語では、構造体のオブジェクト／変数を比較することができませんので、通常の算術型であれば正しい a > b という式はエラーとなります。《意味》が正しいかどうかが調べられるわけです。

このような段階を経て、次は**コード生成**が行われます。すなわち、アセンブリ言語・機械語レベルへの変換が行われます。

さらに必要に応じて**最適化**が行われます。

簡単な最適化の例を考えてみましょう。

```
x = 10;
for (i = 0; i < 1000; i++) {
```

```
        a = 3 * x;
        b = a * i + 5;
        c = a * i + 10;
    }
```

このプログラムにおいて、(プログラム外部からの特殊な技術を用いない限り)、a に代入される値は、必ず 30 となります。したがって、繰り返しの度に 3 * x の計算をするのは馬鹿げた話です。

最適化を行うコンパイラであれば、以下のようなプログラムと同等なコードを生成するでしょう。

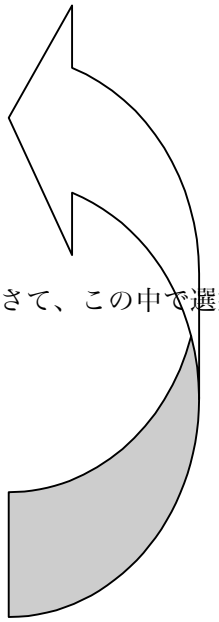
```
    x = 10;
    a = 3 * x;
    for (i = 0; i < 1000; i++) {
        b = a * i + 5;      /* または int __temp = a * i;   y = __temp + 5; */
        c = b + 5;          /*                              z = __temp + 10; */
    }
```

乗算の回数が大幅に減少し、スピードアップが望めます。もちろん、手法としては、高速化を目的とした最適化だけではなく、記憶域を節約するような最適化なども存在します。

さて、電卓が受け付ける計算式の構文は、BNF（Backus-Naur Form）で書かれています。バッカス＝ナウア記法の名前くらいは知っていますよね？　…（沈黙）…

C言語の文は以下のように定義されています。

文　：　名札付き文
　　複合文
　　式文
　　選択文
　　繰返し文
　　分岐文



これは『文』とは、以下のいずれかですよという意味です。さて、この中で選択文は、次のように定義されています。

選択文　：　**if** 文
　　switch 文

そして、**if** 文は

if 文　：　if (式) 文
　　if (式) 文 else 文

と定義されています。この中で登場する『文』とは …
　　… と再帰的に定義されていることに注意しましょう。

奥深いものを感じませんか？

■■ 電卓プログラム ■■

B.Stroustrup『プログラミング言語Ｃ++』から電卓プログラムを示す。
この電卓が受け付ける言語の文法は、以下の通りである。

```
program:
    END                     // ENDは入力の末尾
    expr_list END

expr_list:
    expression PRINT        // PRINTはセミコロン
    expression PRINT expr_list

expression:
    expression + term
    expression - term
    term

term:
    term / primary
    term * primary
    primary

primary:
    NUMBER
    NAME
    NAME <= expression
    -primary
    (expression)
```

実行例

```
r = 2.5
2.5
```

```
//------------------------------------------------------------------------------//
// 第６章 関数   6.1  電卓プログラム                                              //
// p.147  - 158                                                                 //
//------------------------------------------------------------------------------//

#include  <iostream> // 入出力
#include  <string>       // 文字列
#include  <map>          // map
#include  <cctype>       // isalpha()など
#include  <sstream>

using namespace  std;

enum Token_value {
    NAME,          NUMBER,        END,
    PLUS  = '+',   MINUS  = '-', MUL = '*', DIV = '/',
    PRINT = ';',   ASSIGN = '=', LP  = '(', RP  = ')'
};
```

```
int    no_of_errors;
double number_value;
string string_value;
Token_value curr_tok=PRINT;
map<string, double> table;
double expr(bool get);                    //-- 追加 by B.Shibata

double error(const string& s)
{
    no_of_errors++;
    cerr << "error: " << s << '¥n';
    return 1;
}

Token_value get_token()
{
    char ch;

    do {   // '¥n'以外の空白を読み飛ばす
        if (!cin.get(ch)) return curr_tok = END;
    } while (ch != '¥n' && isspace(ch));

    switch (ch) {
    case 0:
        return curr_tok = END;

    case ';':
    case '¥n':
        return curr_tok = PRINT;

    case '*':
    case '/':
    case '+':
    case '-':
    case '(':
    case ')':
    case '=':
        return curr_tok=Token_value(ch);

    case '0': case '1': case '2': case '3': case '4':
    case '5': case '6': case '7': case '8': case '9':
    case '.':
        cin.putback(ch);
        cin>>number_value;
        return curr_tok = NUMBER;

    default:      // NAME, NAME=と表記しなければエラー
        if (isalpha(ch)) {
```

```
            string_value = ch;
            while (cin.get(ch) && isalnum(ch)) string_value.push_back(ch);
            cin.putback(ch);
            return curr_tok=NAME;
        }
        error("bad token");
        return curr_tok=PRINT;
    }
}

double prim(bool get)        // 一次式の処理
{
    if (get) get_token();

    switch (curr_tok) {
    case NUMBER:             // 浮動小数点数
    {   double v = number_value;
        get_token();
        return v;
    }
    case NAME:
    {   double& v = table[string_value];
        if (get_token() == ASSIGN) v = expr(true);
        return v;
    }
    case MINUS:                  // 単項のマイナス
        return -prim(true);
    case LP:
    {   double e = expr(true);
        if (curr_tok != RP) return error(") expected");
        get_token();         // ')'を削除
        return e;
    }
    default:
        return error("primary expected");
    }
}

double term(bool get)    // 乗算、除算
{
    double left = prim(get);

    for (;;)
        switch (curr_tok) {
        case MUL:
            left *= prim(true);
            break;
        case DIV:
            if (double d = prim(true)) {
```

```
                left /= d;
                break;
            }
            return error ("divide by 0");
        default:
            return left;
        }
}

double expr(bool get)    // 加算、減算
{
    double left = term(get);

    for (;;)
        switch (curr_tok) {
         case PLUS:
            left += term(true);
            break;
         case MINUS:
            left -= term(true);
            break;
        default:
            return left;
        }
}

istream* input;

int main(int argc, char* argv[])
{
    switch (argc) {
    case 1:                                  // 標準入力から読み込む
        input = &cin;
        break;
    case 2:                                  // 引数文字列から読み込む
        input = new istringstream(argv[1]);
        break;
    default:
        error("two many arguments");
        return 1;
    }

    table["pi"]= 3.1415926535879323845;        // 定義済み名の挿入
    table["e"]= 2.7182818284590452354;

    while (*input) {
        get_token();
        if (curr_tok == END) break;
        if (curr_tok == PRINT) continue;
```

```
        cout << expr(false) << '¥n';
    }

    if (input != &cin)delete input;
    return no_of_errors;
}
```

自由課題

この電卓プログラムを拡張しよう。

第 11 回

平成 13 年 6 月 28 日

配布資料：23 枚

- 『情報応用特論Ⅰ 講義ノート』　pp.26〜32　(7 枚)
- 『アルゴリズムとデータ構造 第4章』　pp.1〜6　(6 枚)
- 『ソフトウェア工学』　pp.36〜41　(6 枚)
- 『C++によるオブジェクト指向』　pp.28〜31　(4 枚)

--- 昨年度の『講義ノート』より抜粋 ----------

C言語で文字列を空にするためには、右プログラムのように、先頭文字にナル文字を代入する方法と、strcpy 関数で空文字列をコピーする方法とがあります。多くのプログラマは後者を使いますが、私は使いません。きちんと理由があります。

左のプログラムを実行してみましょう。

```
s1 = XBC
s2 = ABC
```

と表示されます。しかし、処理系によっては、記憶域を節約するために、同一綴りの文字列リテラルの記憶域を共有します。この場合、ポインタ s1, s2 は、同一の"ABC"を指すことになります。ちなみに、Bolrand C++では、-d オプションをつけてコンパイルすると、そのようになります。ちょっとコンパイルし直して実行してみましょう。そうすると、今度は

```
s1 = XBC
s2 = XBC
```

となってしまいます。

さらに、右のプログラムを実行してみましょう。

```
ABC
XYZ
1234567890
```

と表示されます。しかし、同一文字列を共有する場合は、

ABC ビープ！ XYZ ビープ！ 1234567890 ビープ！

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

int main(void)
{
    char *s1 = "ABC";
    char *s2 = "ABC";

    s1[0] = 'X';

    printf("s1 = %s¥n", s1);
    printf("s2 = %s¥n", s2);

    return (0);
}
```

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

int main(void)
{
    char *s = "¥n";

    *s = '¥a';

    printf("ABC");
    printf("¥n");

    printf("XYZ");
    printf("¥n");

    printf("1234567890");
    printf("¥n");

    return (0);
}
```

となってしまい、改行するつもりの箇所で、ピーピーなってしまいます。"¥n"だけの文字列リテラルは、

```
¥n  ¥0
```

という２文字分の記憶域を占有します。同一綴りの文字列リテラルを別個のものとして扱う処理系であれば、プログラム中の"¥n"ごとに２バイトの領域を消費することになりますし、同一綴りの文字列を共有する処理系であれば、ここに示したような、《いつのまにか中身が書き換えられてしまう》といった困った事態も発生し得えます。

したがって、改行したいのであれば

```
printf(“¥n”);
```

よりも

```
putchar(‘¥n’);
```

の方が素直で安全です。

さて、右のプログラムや下のプログラムを実行してみてください（同一綴りの文字列リテラルを共有するオプション付きでコンパイルしましょう)。

何だか変な実行結果が得られます。その理由は、皆さんに考えていただくことにしましょう。

話を戻しますが、

```
strcpy(s1, “”);
```

は、わざわざ関数呼出しを行って、引数をやり取りするなどのオーバヘッドがあるだけでなく、いろいろな危険が潜在していることが分かりましたか？

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

int main(void)
{
    char s1[] = "ABC";
    char s2[] = "DEF";
    char *s = "";

    *s = 'X';

    strcpy(s1, "");
    strcpy(s2, "");

    printf("s1 = %s¥n", s1);
    printf("s2 = %s¥n", s2);

    return (0);
}
```

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

int main(void)
{
    char s1[] = "ABC";
    char s2[] = "DEF";
    char *s = "";

    strcpy(s1, "");

    *s     = 'X';
    *(s+1) = '¥0';

    printf("s1 = %s¥n", s1);
    printf("s2 = %s¥n", s2);

    return (0);
}
```

第12回

平成13年7月5日

配布資料：28枚

- 『情報応用特論Ⅰ講義ノート』 pp.33〜34 (2枚)
- 『アルゴリズムとデータ構造 第4章』 pp.7〜9 (3枚)
- 『プログラミング言語Ⅱ』 pp.52〜53 (2枚)
- 『情報基礎ゼミナール』 pp.18〜25 (8枚)
- 『インターネット入門』 pp.20〜22 (3枚)
- 『文字の入力法』 pp.15 (1枚)
- 『ソフトウェア工学』 pp.42〜46 (5枚)
- 『C++によるオブジェクト指向』 pp.32〜35 (4枚)

昨年は、アルゴリズムのほかにも、Javaや Windowsプログラミングもやりました。本年度は、なぜかあまり進みませんでした。はい。

第13回

平成13年7月12日

配布資料：25枚

- 『情報応用特論Ⅰ講義ノート』 p.35 (1枚)
- 『アルゴリズムとデータ構造 第5章』 pp.1〜8 (8枚)
- 『ソフトウェア工学』 pp.47〜58 (12枚)
- 『C++によるオブジェクト指向』 pp.36〜39 (4枚)