情報応用特論 I

講義ノート

－　２００２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、2002 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。

◆　本資料のプログラムを含むすべての内容は、著作権法上の保護を受けており、著作権者である柴田望洋の許諾を得ることなく、無断で複写・複製をすることは禁じられている。

本資料は、Microsoft 社のワープロソフトウェアである Microsoft Word 2002 を用いて作成した。

★　本講義では、以下に示すホームページ上の、各ドキュメントも参照するので、参考にされたい。

柴田望洋後援会オフィシャルホームページ　http://www.BohYoh.com/

ちょっとテスト

情報工学におけるプログラミングあるいはアルゴリズムの問題としては、あまりにも基本的な問題です。もちろん、一般的な情報系大学院の入試問題レベルよりも低いものです。解いてください。

[1] 要素数が n である配列 a に対して、a[idx]に x を挿入する関数

```
void aryins(int a[], int n, int idx, int x);
```

を作成せよ。もちろん、挿入に伴って a[idx], a[idx + 1], … , a[n – 2]を a[idx + 1], a[idx + 2], … , a[n – 1]にずらすこと。

例として、aryins(a, 6, 3, 7);と呼び出した場合を以下に示す。

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
呼出し前	32	55	10	25	72	66
呼出し後	32	55	10	7	25	72

↑
ここに 7 を代入して、以降をずらす。

[2] 要素数が n である配列 a の要素の中で、x 以上 y 以下である全要素を、先頭から順に配列 b に格納し、格納した要素数を返す関数。

```
int arypartcpy(int a[], int n, int x, int y, int b[]);
```

を作成せよ。

例として、arypartcpy(a, 6, 20, 60, b);と呼び出した場合を以下に示す。

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
a	32	55	10	25	72	66
b	32	55	25			

↑
20 以上 60 以下の要素を順に格納して 3 を返す。

[2] 要素数が n である配列 a を単純挿入ソートする関数を作成せよ。

```
void sisort(int a[], int n);
```

第１回

平成 14 年 4 月 9 日

配布資料：

■ 『情報応用工学特論Ⅰ』　pp.0〜2

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)。

まず前ページの問題 (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。問題はＣ言語ですが、作成するのは、ＣでもＣ＋＋でも Java でも構いません。

[1]と[2]は、特定の分野に限られることなく、いろいろな局面で必要となるような関数です。実験データを処理するためのプログラム、事務処理的なプログラム、論理的なプログラムなどです。

また、[3]は、あまりにも基本的なアルゴリズムであり、このくらいは、名前もアルゴリズムも覚えておかなければなりません。

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？

私が、ソフト会社の社長であれば、はっきり言って、この程度の問題を解けない学生は採用しません。そもそも、この問題は、簡単であるばかりでなく、その仕様までもが与えられています。すなわち、どのような作成すべきかが、ほぼ具体的です。

実際に、プログラムを作成するときは、仕様は与えられません。請負ソフトを作るのであれば、顧客の要望を聞き出し、それを具体化し、設計する必要があります。このあたりのことは、昨年度のソフトウェア工学で教えていますね。

※実際には、設計者とプログラム作成者が同一であるかどうかは別ですが。

本学の情報工学科の学生は、あまりにもソフト力が弱いようです。みなさんの研究の専門とは無関係に、この程度の問題は、即、解けるようにならなければなりません。

数年前、学部の３年生が、他大学の大学院を受験したいので、その過去問題の解答を教えて欲しいとしいって相談に来ました。内容は、オペレーティングシステムの内部に関するものや、線形リスト、複雑な文字列処理など。

みなさん、解けますか。

一度、ホームページなどで、他大学の情報系の大学や大学院のカリキュラムを見てください。

さて、《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。

第２回

平成 14 年 4 月 23 日

配布資料：13 枚

- ■　『情報応用特論Ⅰ　講義ノート』　pp.3〜4　（2 枚）
- ■　『C言語によるアルゴリズムとデータ構造』　表紙／p.1,pp.14〜17　（6 枚）
- ■　『情報工学ゼミナール 改』　表紙／pp.0〜4　（5 枚）

情報工学ゼミナール 改

以下に示すプログラムは、読み込んだ変数の値の符号を判定・表示するプログラムです。網掛けの判定部は省略しても正しく動作します。この判定は、必要なのでしょうか、不要なのでしょうか、どちらでもいいのでしょうか？

```
#include  <stdio.h>

int main(void)
{
    int  vx;

    printf("整数値を入力してください：");
    scanf("%d", &vx);

    if (vx == 0)
        puts("その数は０です。");
    else if (vx > 0)
        puts("その数は正です。");
    else if (vx < 0)
        puts("その数は負です。");

    return (0);
}
```

プログラムの流れが、最後の判定である else if (vx < 0)に到達するのは、それ以前の判定である if (vx == 0)と else if (vx > 0)の両方ともに“引っかからなかった”ときだけです。したがって、vx の値が負である場合にのみ、その判定が行われることになりますが、そのとき vx < 0 は必ず成立します。

すなわち、明らかに成立することが分かっている条件判定を、念のために“わざわざ”行っているともいえます。

したがって、網掛け部を取り除いたほうがよさそうだ（これを「別解A」と呼ぶことにします）。

ちなみに、プログラム中の if 文は、以下のようにも書くことができます（これを「別解Ｂ」と呼ぶことにします)。

```
if (vx == 0) puts(″その数は０です。″);
if (vx >  0) puts(″その数は正です。″);
if (vx <  0) puts(″その数は負です。″);
```

これだと、vx の値に関わらず、必ず条件判定が 3 回行われることになります。非常に効率が悪く、論外ですね。

三つの手法の判定回数をまとめると次のようになります。

０／正／負を判断する if 文の三つの実現と判定回数

	０のとき	正のとき	負のとき
プログラム	1	2	3
別解Ａ	1	2	2
別解Ｂ	3	3	3

さて、以下に示す if 文を見比べてみましょう。

```
if (v1 == v2)
    処理Ａ
else if (v1 < v2)
    処理Ｂ
else
    処理Ｃ
```

```
if (vx == 1)
    処理Ａ
else if (vx == 2)
    処理Ｂ
else if (vx == 3)
    処理Ｃ
```

左側の if 文では、プログラムの流れが三つに分岐し、処理Ａ，処理Ｂ，処理Ｃのいずれか１つの“処理”が必ず実行されます。

右側の if 文は、プログラムの流れを三つに分岐し、処理Ａ，処理Ｂ，処理Ｃのいずれかの“処理”が行われるという点で同じように見えますが、vx の値が 1, 2, 3 でない場合は、いずれの“処理”も行われないという点で大きく異なります。したがって、最後の判定を省略することはできません。プログラムの流れは、実質的には四つに分岐しているのです。

「それ以上の分岐がないこと」を明示するためにも、左側のようなケースでは、最後の判定は書かない方が良いといえるでしょう。

もっとも、プログラムの読み手に対して、最後の判定を行うことを強く訴えたいときもあるだろう。“v1 が v2 より大きい場合は、こんなことをやるんだよ。”と、どうしても強調したいのであれば、

```
else if (v1 > v2)
```

と書いても構わないでしょう。おそらくコンパイラの最適化によって、最後の判定は省略されるでしょうから。

Ver.3 のプログラムは、乱数を使っていますので、ゲームらしくなります。

乱数を発生させるのが **rand** 関数です。乱数を発生させるといっても、無から有は生まれません。乱数は、種（*seed*）を元に発生します。したがって、種の初期値が同一であれば、生成される乱数も同一になります。そこで、種の値を変更するのが **srnad** 関数です。

これらの関数の仕様や、乱数の発生法、種の変更方法などについては、柴田望洋後援会オフィシャルホームページ

http://www.BohYoh.com/

を御覧ください。

C言語によるアルゴリズムとデータ構造

アルゴリズムに関しては、難解な本が多いようです。ここでは、私ならではの解説を行っていきます。

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 max3(int a, int b, int c)
{
     int max;

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

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られるものなのですが。一見簡単そうな問題も、実は難しいのかもしれません。

この問題は、来週までの宿題といたします。

第３回

平成 14 年 4 月 30 日

配布資料：15 枚

- 『情報応用特論Ⅰ 講義ノート』　pp.5〜7　(3 枚)
- 『ソフトウェア工学』　表紙／p.1〜9　(10 枚)
- 『情報工学ゼミナール 改』　表紙／pp.5〜6　(2 枚)

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、四つの実現法を示します。

```
/*
    三値の中央値を求める
*/

#include  <ctime>
#include  <iostream>

using namespace    std;

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

//--- Version 0 ---//
int med0(int a, int b, int c)
{
    if (a >  b  &&  b >  c) return (b);
    if (a >  b  &&  b == c) return (b);
    if (a >  c  &&  c >  b) return (c);
    if (a == c  &&  c >  b) return (c);
    if (c >  a  &&  a >  b) return (a);
    if (a == b  &&  b >  c) return (b);
    if (a == b  &&  b == c) return (b);
    if (c >  a  &&  a == b) return (a);
    if (b >  a  &&  a >  c) return (a);
    if (b >  a  &&  a == c) return (a);
    if (b >  c  &&  c >  a) return (c);
    if (b == c  &&  c >  a) return (c);
    if (c >  b  &&  b >  a) return (b);
}

//--- 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[] = {med0, med1, med2, med3};

    for (int i = 0; i < 4; 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。

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、それらに対して、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みなすことができます。

Version 0 は、最もドンくさい方法です。13 パターンの条件をだらだらと列挙しています。

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。

■等値や論理演算の回数は皆さんで比較してみましょう。

■実行時間を比較しましょう。コンパイルオプションによっても変わるかもしれません。

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。

Version 0

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 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													

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。

第４回

平成 14 年 5 月 7 日

配布資料：29 枚

- ■　『ソフトウェア工学』　表紙／p.10〜17　(8 枚)
- ■　『C++によるオブジェクト指向』　表紙／p.1〜12　(13 枚)
- ■　『情報工学ゼミナール 改』　表紙／pp.7〜12　(2 枚)
- ■　『C 言語によるアルゴリズムとデータ構造』　表紙／p.40〜45　(6 枚)

情報工学ゼミナール 改

p.11 は、テロップのプログラムです。たとえば、文字列が"BohYoh"の 6 文字であれば、

```
BohYoh          0 1 2 3 4 5
ohYohB          1 2 3 4 5 0
hYohBo          2 3 4 5 0 1
YohBoh          3 4 5 0 1 2
ohBohY          4 5 0 1 2 3
hBohYo          5 0 1 2 3 4
```

を繰返し表示します。右側に示しているのは、表示する文字の添字です。0 を先頭に表示して、1 を先頭に表示して、…、5 を先頭に表示すると、最初に戻って 0 を先頭に表示しることになります。

それでは、テロップの流れを逆にしてみましょう。次のように表示すればよいわけです。

```
BohYoh          0 1 2 3 4 5
hBohYo          5 0 1 2 3 4
ohBohY          4 5 0 1 2 3
YohBoh          3 4 5 0 1 2
hYohBo          2 3 4 5 0 1
ohYohB          1 2 3 4 5 0
```

ということは、先頭文字のローテーションを 0→1→2→3→4→5 だったのを、0→5→4→3→2→1 にすればよいですね。

したがって、次のように変更するだけでいいのです。

```
if (cnt < name_len - 1)          →      if (cnt > 0)
    cnt++;                                  cnt--;
else                                    else
    cnt = 0;                                cnt = name_len - 1;
```

“コンパイラ”について

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;
```

まず、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て変数値を変更しない限り)、`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。

第５回

平成 14 年 5 月 14 日

配布資料：29 枚

- 『情報応用特論Ⅰ 講義ノート』　pp.8〜13　(6 枚)
- 『ソフトウェア工学』　表紙／p.18〜20　(3 枚)
- 『情報工学ゼミナール 改』　表紙／pp.13〜15　(3 枚)
- 『C++によるアルゴリズムとプログラミング』　表紙／p.1〜3　(4 枚)

表示する数値の桁数を変数で表示

右に示すプログラムは、1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, … を 1 桁ずつずらしながら表示する C++プログラムです。

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。

それでは、これをＣ言語で実現してみたものを二つ示します。

左下に示すのが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。

もう一つの実現例を右に示します。

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

int main(void)
{
    for (int i = 1; i <= 15; i++)
        cout << setw(i) << i % 10  << '¥n';

    return (0);
}
```

```
1
 2
  3
   (中略)
         9
```

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）。

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、ここで使っている*です。

```
#include <stdio.h>

int main(void)
{
    int i;

    for (i = 1; i <= 15; i++)
        printf("%*d¥n", i, i % 10);

    return (0);
}
```

```
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i <= 15; i++) {
        for (j = 1; j <= i; j++)
            putchar(' ');
        printf("%d¥n", i % 10);
    }
    return (0);
}
```

したがって、この **printf** 関数の呼び出しは、

『**i % 10** 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るのですね。

第６回

平成 14 年 5 月 21 日

配布資料：11 枚

- 『情報応用特論Ⅰ　講義ノート』　pp.14　(1 枚)
- 『ソフトウェア工学』　表紙／p.21〜23　(3 枚)
- 『情報工学ゼミナール 改』　表紙／pp.16〜20　(5 枚)
- 『C++によるアルゴリズムとプログラミング』　表紙／p.4〜5　(2 枚)

```
/*
    赤・青・黄を交互に表示（その１）
*/

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

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    clock_t s = clock();
    clock_t c;
    do {
        if ((c = clock()) == (clock_t)-1)       /* エラー */
            return (0);
    } while (1000UL * (c - s) / CLOCKS_PER_SEC <= x);
    return (1);
}

int main(void)
{
    int  i = 0;

    while (1) {
        switch (i) {
         case 0: printf("¥r赤");  i++;  break;
         case 1: printf("¥r青");  i++;  break;
         case 2: printf("¥r黄");  i=0;  break;
        }
        sleep(1000);
    }

    return (0);
}
```

左に示しているのは、『赤』・『青』・『黄』を交互に 1 秒ずつ、同一行に表示する、何の変哲もないプログラムです。

さて、このプログラムでの表示順序を『赤』・『桃』・『黄』・『青』に変更してみましょう。そうすると、プログラムの **while** 文を以下のように書き換えなければなりません。

その過程で、プログラムの一部をコピーしたり移動したりし、さらに手作業で i++ と i=0 を間違えないように交換することになります。

```
while (1) {
    switch (i) {
     case 0: printf("¥r赤");  i++;  break;
     case 1: printf("¥r桃");  i++;  break;
     case 2: printf("¥r黄");  i++;  break;
     case 3: printf("¥r青");  i=0;  break;
    }
    sleep(1000);
}   return (1);
```

そこで、表示する文字列を、文字列の配列（char 型の２次元配列）に格納することにしましょう。そうすると、プログラムの main 関数を以下のように変更することになります。

```
int main(void)
{
    int  i = 0;
    char cstr[][3] = {"赤", "桃",  "黄", "青"};

    while (1) {
        printf("¥r%s", cstr[i]);
        if (++i > 3) i = 0;

        sleep(1000);
    }

    return (0);
}
```

このように実現していれば、配列 cstr の初期化子を適当に変更するだけでいいですね。… といいたいのですが、もう一箇所残っています。

それは網掛けをしている 3 です。ここは、表示する文字列の個数、すなわち、配列 cstr の要素数から 1 を減じた値でなければなりません。したがって、さらに文字列を一つ増やして、5 個にするのであれば、この値を 4 に変更しなければなりません。

したがって、配列の要素数もプログラムに自動的に計算させましょう。そのプログラムを示します。

```
int main(void)
{
    int  i = 0;
    char cstr[][3] = {"赤", "桃",  "黄", "青", "白"};
    int  num = sizeof(cstr) / sizeof(cstr[0]);

    while (1) {
        printf("¥r%s", cstr[i]);
        if (++i > num - 1) i = 0;

        sleep(1000);
    }

    return (0);
```

これで、文字列の変更に伴って、プログラムを手作業で更新する必要がなくなります。

さて、要素数の求め方について、より単純な 1 次元配列で確認しましょう。

右に示すプログラム部分を例に考えます。

```
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、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。

--- 昨年度の『講義ノート』より抜粋 ----------

キーボードタイピングについて

□ 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、皆さんは、正しい指使いを知っていますか？ 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。

□　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。

『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。

『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、姶　…　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）。

第７回

平成 14 年 5 月 28 日

配布資料：32 枚

■	『情報応用特論Ⅰ　講義ノート』	pp.15〜20	(6 枚)
■	『ソフトウェア工学』	表紙／p.24〜31	(8 枚)
■	『情報工学ゼミナール 改』	表紙／pp.13〜17	(5 枚)
■	『C++によるアルゴリズムとプログラミング』	表紙／p.184〜185	(2 枚)
■	『秘伝Ｃ言語問答ポインタ編第２版』	表紙／p.305〜315	(11 枚)

『C++によるアルゴリズムとプログラミング』

ある程度以上の規模のプログラムは、複数のソースプログラムに分けて開発するのが常識です。

『秘伝Ｃ言語問答ポインタ編第２版』

関数へのポインタを利用することによって、プログラムの開発時ではなく、プログラム実行時に呼び出す関数を決定できる、柔軟で汎用性の高いプログラムが作成できます。

通常の関数呼出し	関数へのポインタを使った呼出し
f()　← 関数 f を呼び出す コンパイル時に、どの関数を呼び出すかが決定される。	fp()　← fp が指している関数を呼び出す コンパイル時ではなく、実行時に、どの関数を呼び出すかが決定される。

--- 昨年度の『講義ノート』より抜粋 ----------

『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。

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）。

□　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。

□　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。

『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。

第８回

平成 14 年 6 月 4 日

配布資料：32 枚

■	『情報応用特論Ⅰ 講義ノート』	pp.21〜22	(2 枚)
■	『情報工学ゼミナール 改』	pp.21〜25	(5 枚)
■	『C++によるアルゴリズムとプログラミング』	p.6〜12	(7 枚)
■	『C++によるオブジェクト指向』	p.18〜21	(4 枚)
■	『C++によるアルゴリズムとプログラミング』	p.121〜127	(7 枚)

--- 昨年度の『講義ノート』より抜粋 ----------

「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。

《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";
```

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）。

『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。

第９回

平成 14 年 6 月 11 日

配布資料：32 枚

- ■　『情報応用特論Ⅰ　講義ノート』　p.23　(1 枚)
- ■　『情報工学ゼミナール 改』　pp.26〜27　(2 枚)
- ■　『ソフトウェア工学』　p.34〜39　(6 枚)
- ■　『C++によるオブジェクト指向』　p.22〜26　(5 枚)
- ■　『C++によるアルゴリズムとデータ構造』　p.128〜137　(10 枚)

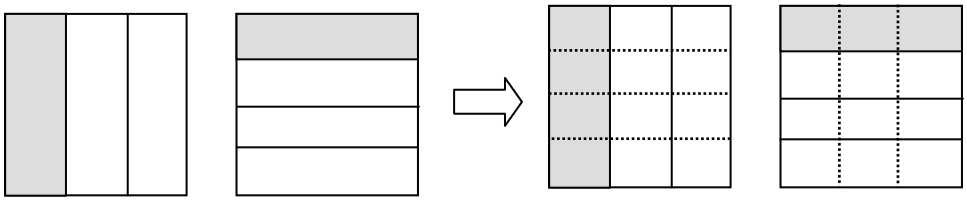
--- 昨年度の『講義ノート』より抜粋 ----------

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）。



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。

```
５個の整数を入力せよ。
No.1：52
No.2：1
No.3：74
No.4：18
No.5：11
50以上80未満は2個です。
No.1=52
No.3=74
```

このプログラムの実現例を示します。左側はＴ君、右側が私のものです。

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。

```c
#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);
}
```

```c
#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);
}
```

第１０回

平成 14 年 6 月 18 日

配布資料：32 枚

- ■　『情報応用特論Ⅰ　講義ノート』　pp.24〜25　(2 枚)
- ■　『情報工学ゼミナール 改』　pp.28〜31　(4 枚)
- ■　『C++によるオブジェクト指向』　pp.27〜36　(10 枚)
- ■　『C++によるアルゴリズムとデータ構造』　pp.138〜149　(12 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);
}
```

第１１回

平成 14 年 6 月 25 日

配布資料：32 枚

- 『情報応用特論Ⅰ　講義ノート』　p.26　(1 枚)
- 『情報工学ゼミナール 改』　pp.33〜38　(6 枚)
- 『ソフトウェア工学』　pp.40〜46　(7 枚)
- 『C++によるアルゴリズムとプログラミング』　pp.13〜18　(6 枚)
- 『C 言語によるアルゴリズムとデータ構造』　p.151〜163　(13 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。

```
#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);
}
```

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。

```
#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);
}
```

```
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！

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？

```
#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);
}
```

```
#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 14 年 7 月 4 日

配布資料：32 枚

■ 『情報応用特論Ⅰ 講義ノート』	p.27〜28	(2 枚)
■ 『情報工学ゼミナール 改』	pp.39〜41	(3 枚)
■ 『ソフトウェア工学』	pp.47〜53	(7 枚)
■ 『C 言語によるアルゴリズムとデータ構造』	pp.164〜183	(20 枚)
■ 『C++によるオブジェクト指向』	pp.37〜43	(7 枚)

南郷継正『武道講義第一巻／第二巻』より引用

人間から誇りを除いたならば、残りうるものはただに動物としてのヒトだけでしょう。

人間が誇りを失ったならば、後に残りうるのは実体としては本能レベルの猿的人間だけでしょう。

それだけに、いかなる大学であれ、いかなる学生であれ、その最高レベルにふさわしく自らに誇りを把持し続けなければならないし、その誇りが嘘にならないように自らの努力と情熱で把持しつづけることが肝心でありましょう。そうであるべきなのに、人間たるものが自らの意志でむざむざ誇りを棄ててよいものではありません。

ですからそのためにも、ことさらに学としての弁証法や学としての認識論についても、誇り高くあるために、初心のうちに誇りをもってしっかりと学習させておく必要があろうというものであります。

（中略）

彼ら若き学徒は、教科書や参考書をとおして事実的に存在する理論や論理を知ることはあっても、それらを論理的研鑽として学習する機会は、少なくとも文部省検定下における学校教育としてはもっていないからです。つまり、学校では大学を含めまして、それらを知識としては教えてくれても、論理的学習としては学ばせてはくれません。これは大学教官をも含めてほとんどの先生の実力がないからでもあります。

それだけに彼ら若き学徒は、心では教官を軽蔑しながら、しかたなくただにそれらを自らの興味として自学自習するしかないわけです。しかしそのばあいとて、参考書なるものも自らの興味で選ぶしかないばかりか、その書物たるものはインチキ同然のものがハバをきかせている現実ですので、私のように見事なる偶然さで世界最適の基本書・三浦つとむ著『弁証法はどういう科学か』（講談社）に出逢うといった者は数えるくらいしかいません。

（中略）

通常の人間の受験期は、その人にとっては、教育という名の文化遺産の習得形式に関わってからの最大の成長期とともに存在することになるものである。かくのごとき現実は、わが日本の教育の現状からはまずは例外はない、ととってよいと思う。

（中略）

体が成長期にあるということは、実体が実体として直接に発育・発達することは当然的であるが、ここを即物的実体論的にとらえてはならない。

（中略）

思春期（中学生）の実体＝五感器官の発育・発達は急速かつ激的な独立的・連動的・相互浸透的であると説いたが、これは少し大仰的には、眼前での花の香り一つにたいしても、である。

（中略）

その細胞分裂は小学生の大まかな、ゆるやかなと違い、あるいは大人のほとんど変化のないものと大きく異なり、細胞分裂に関わる細胞そのものが急速に分裂するだけでなく、激的分裂をなしとげていくということである。もっと説けば、この細胞分裂は小学生の成長レベルたる細胞分裂ではなく、大人への脱皮たる細胞分裂であり、思春期たる特殊性を恥じする細胞分裂なのである。あえて述べれば細胞分裂の質（形式でない！）が違うのである。子供の細胞が大人への脱皮へ向けた思春期の細胞へと質的に大きく、激的に変化するのである。

（中略）

これはひとえに、思春期における細胞分裂を含む細胞全体のそしてそれらを統括する脳細胞の急的かつ激的なる質的変化にもとづく発展によるものであり、かかることが、花の香りにたいしてばかりでなく、すべての事物・事象に関わって生成していくのが実体たる思春期・感覚器官であり、あるいはその機能たる感覚である。

（中略）

その間、この中学生は眼をつむっているとしておく。その花の香りを胸一杯に吸い込んでは味わい、吸い込んでは味わいをくり返している途上も、この嗅覚器官は思春期独特の成長過程、すなわち、急的・激的なる質の深まった細胞分裂をくり返しているのである。それだけに、ここに一つの二重構造が現象していることに気がつく必要がある。

（中略）

少し、詳論するならば、中学生（思春期）は自らの自然成長的成長に加うるに、自らの労働による目的意識的な成長をもってして、そこに加速度的な量質転化化をもたらすべく、相互浸透を行なっているのにたいし、大人は自らの無成長なる実体を目的意識的な労働でなんとか実体的な成長を果たしたいとの願望をもったにしても、それは幻にしかすぎないとの構造が横たわっててるということである。

（中略）

人間の認識は大きく二つにわける必要性があるだけに秀れて教育の問題となるのです。すなわち、教育制度の必然性が浮上してきたのです。

認識というのは、よく、対象の反映として成立するといわれます。これは少し乱暴にいいますと、人間は対象の反映があって行動し、反映を抜きにした行動はないということで

す。ですから、厳しい言葉を用いますと、人間を考えるばあい、本能を主体に考えては駄目だ、ということなのです。本能の統括すらも反映が主体になっていく、ということです。たとえば、赤ちゃんを考えてみましょう。赤ちゃんは本能の統括では生きていけません。猫ですと、生まれてすぐに本能にもとづいて自分で、教わることなくお母さんのオッパイを吸います。ところが人間の赤ちゃんは、ただ泣くだけです。お母さんがオッパイを赤ちゃんの口元へもっていってやらないと駄目なのです。

これでわかるように、人間の赤ちゃんは本能では駄目でお母さんが手を貸してやらなければなりません。オッパイをあまり吸わない赤ちゃんをみて、「この子は食が細い」と思うべきではなく、必要なだけは無理をしても吸わせることが大事なのです。ここをないがしろにすると、赤ちゃんは怠けることを覚えるのです。労働しなくてもよい、ということを覚え（反映）ます。

赤ちゃんの乳を吸う力は吸わせることによってついてくるものです。ですから、吸う！ということをまともに覚えさせない哺乳ビンは赤ちゃんをいわば怠けさせるものです。哺乳ビンの欠点は、栄養だけの問題ではなく、この労働＝まじめに吸わないとオッパイがでないということの欠如にもあるのです。哺乳ビンでは怠けてもでてくるものですから。したがって全力で努力するということを覚える機会を逃がすことでもあるのです。

もう一つの欠点は反映の違いが大きすぎて結果的にも、過程的にも感性の育ち方が違ってくるということです。オッパイと哺乳ビンとでは、スルメとチューイング・ガム以上の差があり、赤ちゃんに育つ心の深みが違ってきます。ここを、「そんなことを！」と馬鹿にしてはいけません。哺乳ビンのゴムはどうしてもオッパイの柔らかさにはかないませんし、感触が大きく違います。それにオッパイはいつまでも温かいのにたいし、哺乳ビンは冷えていきます。これまた当然に赤ちゃんの感性＝認識に大きな差となってあらわれるのです。オッパイには本物の深みがあり、あったかみがあります。それが赤ちゃんに反映し、その認識ができあがっていくのです。

（中略）

認識は対象が脳細胞に反映してできます。これは五感器官（具体的には目・耳・鼻・舌・皮膚です）をとおしての反映です。ですから認識は五感像＝五感感情、つまり、五感器官をとおしてきた五感による浸透した合成像なのです。

（中略）

以上要すれば、思春期においては認識＝感情が極端に増幅され、それが実体を統括してきかねない条件が整っているので、現実の事実・事実以上の反映・問いかけがなされることになるわけです。ここにおいて、「私の親はアホだ！」との像が形成されると、それが突然にすさまじいばかりの嫌悪感として成長して、アレヨアレヨという間もなくの家出、自殺、暴走となりかねない、ということです。

『情報応用工学特論Ⅰ』これにて終了。