プログラミング学習について

風の回廊「風草」

平成 21 年 10 月 29 日

1　はじめに

C 言語によるプログラミングの授業であったコンピューター工学は授業の進度や内容共に，既にプログラミングを行った事のある学生向けになっていたため，内容を理解するのが難しかった．そのため，本レポートは授業で補いきれなかった内容について補完するものとする．ただし，本レポートの読者がどの程度のレベルなのか分からないため，簡単な内容にしたつもりだが，分からないところがあれば適宜質問していただきたい．
※質問があれば本レポートも更新される予定です

1.1　著者のプログラミング学習方法について

私は工業高校出身であり，高校の時に BASIC 言語によるプログラミングを行っていた．当時行っていた効率的だと思える学習方法 (遊び) は，クラスメイトとゲームを作り合ったことである．
ゲームを作るには，作りたいゲームのアルゴリズムを描くことが必要である．ここでいう，アルゴリズムを描くとはフローチャートを描くことと同義である．

そのため，ゲームなどの興味のあるものや実用的なものを使用して学習することが効率的な学習をするために重要である．次頁に，私が高校時代に作成したものを BASIC から C 言語に変更したプログラムとフローチャートを示す．

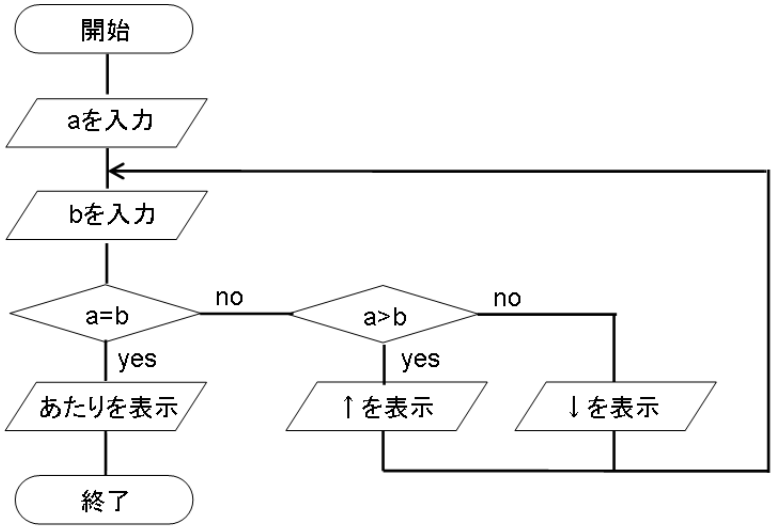

Fig.1: 数当てプログラムのフローチャート

数当てプログラムのソースコード

```
1   #include <stdio.h>
2   main () {
3      int a;
4      int b;
5      scanf("%d",&a);
6      while(1){
7         scanf("%d",&b);
8         if(a == b) {
9            printf("あたり");
10           break;}
11        else if(a > b) {printf("↑");}
12        else if(a < b) {printf("↓");}
13     }
14     return (0);
15  }
```

2　C 言語の学習について

C 言語の学習は授業で使用した明解 C++の本を使用するものとする．なお，授業で使用したスライドも馬鹿に出来ないので良く読むように！

2.1　1 章から 4 章までについて

for 文，if 文，などの 1 章から 4 章までの基礎的な文については既に使えるものとして話を進める．ただし，ここで言う” 使える” とは簡単なプログラムを作成できることであり，複雑なプログラムが作成できることではない．（本当は，演習レベルのプログラムができると良いんですが．出来ない場合は，フローチャートを描く練習も兼ねて何をどのように処理すれば出来るか考えると良いかもしれません．時間はかかってしまうと思いますが，慣れると簡単なフローチャートが頭に浮かぶようになりますよ.）

なお，4 章で改めて覚え直していただきたいのは，P130 からの型変換 (キャスト) である．キャストとは言葉の通り int や double などの型を変換する際に使用するものである．

使用方法としては,「型 （ 式 ）」のように使用する．例を挙げると,「double (x+y) /2.0」などである．

2.2　5 章 (関数) について

関数は，プログラムの部品であり，関数だけではプログラムは動く事ができない．そのため，関数を使用するより先に関数を定義することなどが必要となる (先に定義しない場合はプロトタイプ宣言が必要となる)．ここで，Fig. 2 に関数の例を示す．また，関数を使用する際には仮引数部に引数を与えることができる．ただし，仮引数と引数の型は一致していないとならないという制約 (基本的にはキャストされてコンパイル自体は出来る) も存在する (次頁参照)．Fig. 3 に関数の引数の与え方と関数の使用方法について示す．Fig. 3 のように関数を使用する際は「関数の名前 (引数 1，引数 2，…);」となる．

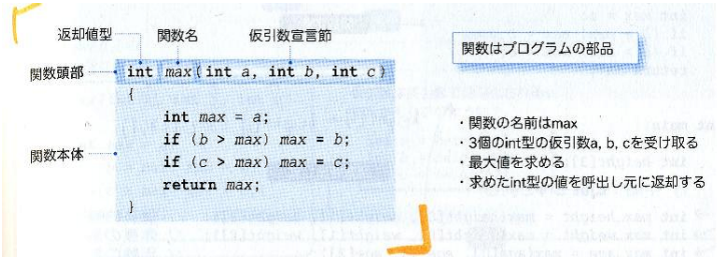

Fig.2: 関数の例 [1]

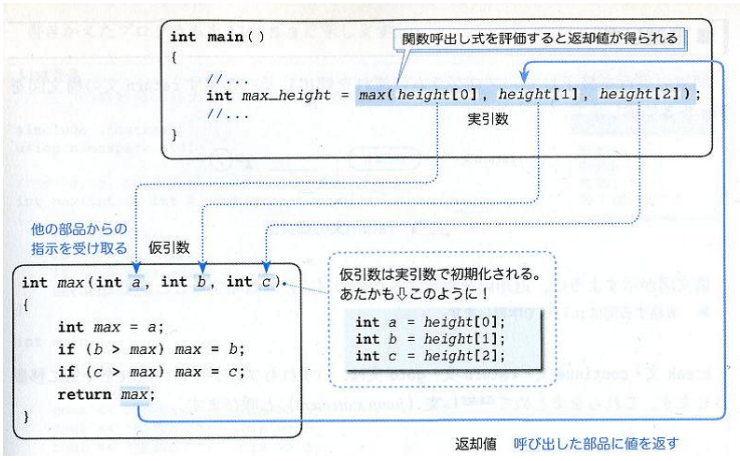

Fig.3: 関数の引数 [1]

ここで，関数の引数について少し注意点を述べておこう．仮引数と引数の型が一致していない場合には Fig. 4 のようにコンパイルは通るが，ワーニングが発生してしまう．それに対して，仮引数と引数の数が一致していない場合には Fig. 5 のようなエラーが発生してしまうため注意が必要である．(引数の注意点のソースコードで説明すると，5 行目が「double test()」となっている場合や，13 行目が「test();」となっている場合である．なお，両方が変更されている場合にはエラーは発生しない．)

```
$ g++ -o kansuu kansuu.c
kansuu.c: In function `int main()':
kansuu.c:13: warning: passing `double' for converting 1 of `double test(int)'
```

Fig.4: 引数の注意点

```
$ g++ -o kansuu kansuu.c
kansuu.c: In function `int main()':
kansuu.c:6: error: too few arguments to function `int test(double)'
kansuu.c:13: error: at this point in file
```

Fig.5: 関数の引数の問題

引数の注意点のソースコード

```
1  #include <cstdlib>
2  #include <iostream>
3  using namespace std;
4
5  double test(int a)
6  {
7  return 0;
8  }
9
10 int main()
11 {
12 double a=5;
13 test(a);
14 return 0;
15 }
```

値渡しと参照渡しについても述べておこう．値渡しは Fig. 6 のように実引数の値が書きかえられる可能性が無いのに対し，参照渡しは Fig. 7 のように実引数の値が書き換えられる可能性がある．そのため，参照渡しは不用意に用いるべきではないと言われている．

通常は値渡しが用いられるが参照渡しを用いる場合は，仮引数の部分に「void swap(int &a,int &b)」のように&を付加することで参照渡しとなる．

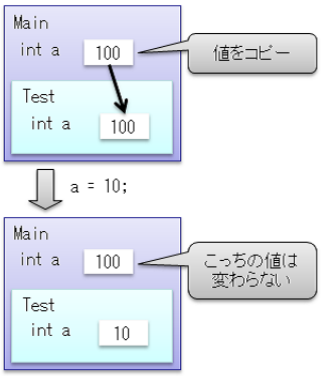

Fig.6: 値渡し [2]

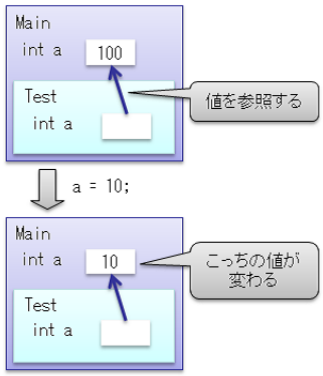

Fig.7: 参照渡し [2]

値渡しのソースコード

```
1  #include <cstdlib>
2  #include <iostream>
3  using namespace std;
4  void swap(int a,int b)
5  {
6 int tmp =a;
7 a=b;
8 b=tmp;
9  }
10  int main()
11  {
12 int a=3,b=8;
13 swap(a,b);
14 cout << "a = " << a <<  endl;
15 cout << "b = " << b <<  endl;
16 return 0;
17  }
```

実行結果は

```
a=3
b=8
```

参照渡しのソースコード

```
1  #include <cstdlib>
2  #include <iostream>
3  using namespace std;
4  void swap(int &a,int &b)
5  {
6 int tmp =a;
7 a=b;
8 b=tmp;
9  }
10  int main()
11  {
12 int a=3,b=8;
13 swap(a,b);
14 cout << "a = " << a <<  endl;
15 cout << "b = " << b <<  endl;
16 return 0;
17  }
```

実行結果は

```
a=8
b=3
```

2.3　6 章 (ポインタ) について

2.3.1　ポインタとは

前章で述べた値渡し，参照渡しは C++言語では対応しているが，c 言語では対応していない．そのため，c 言語では本章で述べるポインタという方法が不可欠となる．ただし，C++言語でポインタを用いることで得られるメリットは，プログラムの高速化やメモリの使用量を減らすことが可能な点もあるが，一番のメリットは構造体を用いることが出来るという点である．

ポインタ (pointer) とは，他のオブジェクトのアドレスを指すオブジェクトであり，オブジェクトのアドレスを取り出す『&オブジェクト』とアドレスのオブジェクトを取り出す『*アドレス』がある．一般的なポインタの使い方を以下に示す．

『&オブジェクト』を用いてアドレスを
表示する方法のソースコード

```
1  #include <cstdlib>
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6 int ken = 170; //宣言は通常と同じ
7  cout<< ken<< endl; //オブジェクトを表示
8  cout<< &ken<< endl;
//『&オブジェクト (ken)』としてアドレスを表示
9  return 0;
10 }
```

『*アドレス』を用いてオブジェクトを
表示する方法のソースコード

```
1  #include <cstdlib>
2  #include <iostream>
3  using namespace std;
4  int main()
5  {
6 int ken = 170;
7 int* ken2 = &ken;
//*をつけてポインタとして宣言し，ken のアドレスを代入する
8  cout<< *ken2<< endl;
//『*アドレス (ken2)』としてオブジェクトを表示
9  cout<< ken2<< endl; //アドレスを表示
10  return 0;
11 }
```

このように，&と*の違いは Table. 1 の関係となっている．

Table 1: &と*の違い

宣言	オブジェクトの出力方法	アドレスの出力方法
int ken=オブジェクト (数値)	ken	&ken
int *ken=アドレス	*ken	ken

また,『int *ken』に 170 の数値を直接代入したい場合は,『int* ken = new int(170)』としてして，明示的に記憶領域を確保しなければならない．それに伴い，ポインタに値を代入したい場合は，以下に示すようなオブジェクトの初期化が必要となる．

```
C++ 教科書 P232
double *taro = new double; // ゴミの値で初期化
double *taro = new double(); // 0 で初期化
double *taro = new double(170); // 170 で初期化
```

つまり，全頁を踏まえてポインタを分かりやすく述べると，オブジェクトを使用する (作成する) ということは名前のついた箱を用意することであると考えてほしい．その沢山の箱の中から「A」の箱,「B」の箱を識別できるのはコンピュータだけであり，私たちにはどれがどれか分かりにくい．そこで,『「A」の箱』という指定方法ではなく,『赤色の箱』と言ったようなアドレス (メモリに割り当てられている住所) を利用する事で私たちでも正確に箱の指定が出来るようになる．これが，ポインタの考えと近いと思われる．(厳密性は考慮していない)

2.3.2 配列とポインタの関係

配列とポインタの関係はある意味同じものといっても間違いはない．ただし，完全に同じではないので注意が必要である．

同じと言う意味は配列の表し方とポインタの表記の仕方が同じという意味である．(配列はは先頭のアドレスから必要要素分のメモリを確保する性質があり，ポインタは記憶領域は別に確保する必要がある) そのため，配列をポインタとして表記することが可能であり，配列の先頭アドレスをポインタとすることで，配列全部をコピーする事も可能である．

例として，以下に示す.
int y[5];
int *p;
p = y;

このような，プログラムの場合配列の先頭アドレス (配列は要素数を書かないと先頭アドレスを出力する) とポインタのアドレスを一致させることでコピーすることができる．Table. 2 と Table. 3 に配列とポインタの関係を示す．(int a[5] = 1， 2， 3， 4， 5} ;int*p = a;　として定義した)

Table 2: 配列とポインタのオブジェクトの表記方法

オブジェクトを表記する方法				出力されるオブジェクトの値
a[0]	*a	*p	p[0]	1
a[1]	*(a+1)	*(p+1)	p[1]	2
a[2]	*(a+2)	*(p+2)	p[2]	3
a[3]	*(a+3)	*(p+3)	p[3]	4

Table 3: 配列とポインタのアドレスの表記方法

アドレスを表記する方法				出力されるアドレスの値
& a[0]	a	p	& p[0]	100 番地
& a[1]	a+1	(p+1)	& p[1]	104 番地
& a[2]	a+2	(p+2)	& p[2]	108 番地
& a[3]	a+3	(p+3)	& p[3]	112 番地

2.4 クラス

クラスとは，A や B，C といった情報をバラバラに扱うのではなく，ひとまとめにしたオブジェクトとして扱うことがクラスの基本的な考え方です.
(クラスについてはあまり深く勉強した事がないので更新するか微妙です)

2.5 構造体

構造体はクラスの中の一つであり，様々な情報をひとまとめとして扱うことができるようになります．ただし，全てのメンバは公開されることとなります (基本的にあまり気にしなくても問題ないです).

使い方としては,

```
struct xyz
        int x;
        long y;
        double z;
;
struct xyz a;
```

のように，構造体の中身に関する定義 (xyz{…} の…の中身) と使用する構造体名の定義 (struct xyz a;) に分かれています．メンバの中の値を読み込むときや，書き込む際には,『a.x=5;』や『c=a.y』といった形で利用します.

a といった要素に速度や角度など様々な情報が入っていた場合などに全てを『a. ○』といった形で用いれるため，構造体は非常に重宝します.

参考文献

[1] 柴田望洋：明解 C++，ソフトバンククリエイティブ株式会社，2006.

[2] 引数の参照渡し（C# によるプログラミング入門）：http://ufcpp.net/study/csharp/sp_ref.html