第二章　配列とポインタ変数

２．１　配列の復習

同じ型を持つ複数個の変数を取り扱うときは、配列を使ってプログラミングする事が多くあります。今回は、配列についてもう少し詳しく述べます。

int 型変数を宣言したとき、メモリの適当な場所に４バイトの記憶領域が確保されることは、前回見ました。それでは、int の配列を宣言したとき、メモリに対してどのような操作が行われるのでしょうか。思い出してみましょう。

復習例題）int 型成分１０の配列のサイズを表示させよ

```
main(){
        int         a[10];
        printf("%d¥n",sizeof(a));
}
```

実行結果

40

練習問題 2.1.1　int 型成分１０の配列 a[10]を宣言し、各成分のアドレス(&a[0]～&a[9]）を表示させよ。

２．２　ポインタ型変数の復習

第一章の内容がすべて理解できていれば、次の問題が簡単に解けます。ポインタ型変数の値に１を足すと、何が起こるか、思い出してください。

練習問題 2.2.1　以下の処理を行うプログラムを作成せよ。

（１）成分１０個の int 型配列　a　を宣言し、適当に値を代入する

（２）int へのポインタ型変数 p に、a の先頭アドレス（&a[0])を代入

（３）＊演算子を用いて、a[0]～a[9]の値を表示させる。

この問題とその答えが、十分理解できるまで、先に進まない方が良いです。理解した方のみ、先に進んでください。

次のプログラムを見てください。

例題）char 型の配列 a を宣言し、 a に文字列 abc を代入し、 a の内容を表示するプログラムを作成せよ。

```
main(){
        char    a[4];
        a[0]='a';
        a[1]='b';
        a[2]='c';
        a[3]='¥0';
        printf("%s",a);
}
```

実行結果

abc

printf("%s",...）は配列名を引数にする関数で、みなさんにもおなじみですね。さて、ここで配列名 a を単体で用いていると言うことは、

配列名 a が、それ自身、何らかの意味を持っている

と言うことです。さて、何の意味を持っているのでしょう。printf の仕様について少し詳しい本を調べると、次のように書いてあります。

printf("%s",...)は char 型へのポインタを引数とする。

ここで char 型へのポインタとは、

char 型へのポインタ型変数もしくはその値（アドレス）

と言う意味です。つまり、大雑把に言ってしまえば

printf(%s"...)　の引数部分...には、char 型変数のアドレスが入る。

ということです。実際のプログラムで確かめてみます。

例題）printf("%s",...)を用いて、char 型変数の内容を表示する。

```
main(){
        char    c;
        c='a';                  /*  c には a のアスキーコードが入る                 */
        printf("%s",&c);  /*   printf(%s"...)　の引数部分には          */
                                /*   char 型変数 c のアドレス&c　が入っている     */
}
```

実行結果

a　拆｜

結果として変数 c に格納された文字 a が表示されました。しかし、なにやらノイズがでています。これは、メモリ上で変数 c の隣の領域に、なにやら値が入っていることを意味しています。

例題）上の例題において、変数 c に隣接する番地に、何が入っているか確かめよ。

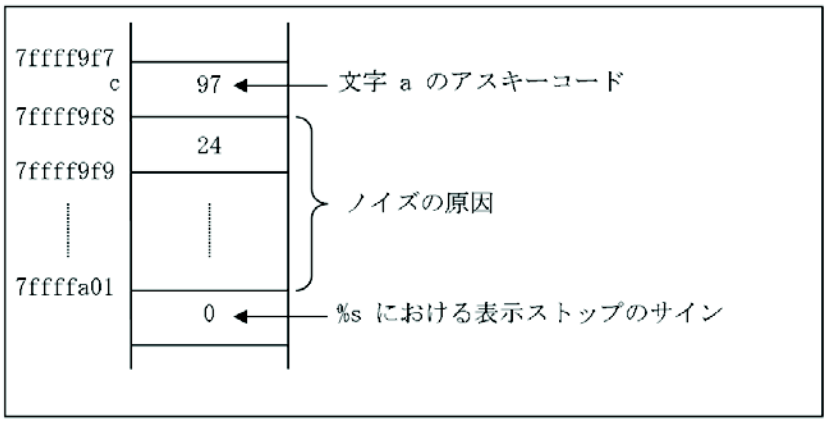
```
main(){
        char    c;
        char    *cp;
        int     i;
        c='a';
        cp=&c;
        for(i=0;i<=19;i++)
        printf("%p %d¥n",cp+i,*(cp+i));     /* cp+i 番地の番地名及び          */
                                            /* そこに記載されている値*(cp+i) */
                                            /* を表示させる                   */
}
```

実行結果

```
0x7ffff9f7 97
0x7ffff9f8 24
0x7ffff9f9 -6
0x7ffff9fa -1
0x7ffff9fb 127
0x7ffff9fc -82
0x7ffff9fd -7
0x7ffff9fe -83
0x7ffff9ff 42
0x7ffffa00 1
0x7ffffa01 0                <---   0='¥0'を思い出せ
0x7ffffa02 0
0x7ffffa03 0
0x7ffffa04 68
0x7ffffa05 -6
0x7ffffa06 -1
0x7ffffa07 127
```

0x7ffffa08 76
0x7ffffa09 -6
0x7ffffa0a -1

先頭の９７は、文字 a のアスキーコードです。後の数字（24,-6,-1 等）は、メモリに最初から入っていた値（ノイズと思ってください）です。これらノイズのうち、表示可能なモノをすべて表示して、0x7ffffa01 番地の 0 を見つけた段階で、printf は表示をストップします。



メモリ上で変数 c の周辺に何が入っているかを表す図

以上の議論から分かることは、

printf("%s"...)の引数部には char 型変数のアドレスが入る。

printf("%s"...)は、引数として入力されたアドレスから順に

メモリを 1 バイトずつに区切り、 0 (='¥0')が現れる手前までを表示する。

ということです。話が脱線してしまいましたが、先に述べた

配列名 a は何を意味するか

という質問の答えは、読者のみなさんにはもう分かったことでしょう。

配列名 a は、配列 a の先頭アドレス&a[0]を意味します。

Ｃ言語プログラムの中では、ほとんどすべての場合、

配列名＝先頭成分のアドレス

と読み替えられます。このことを次の例題で再確認します。

例題)

```
main(){
        char      a[10];
        printf("%p %p¥n",&a[0],a);
}
```

実行結果

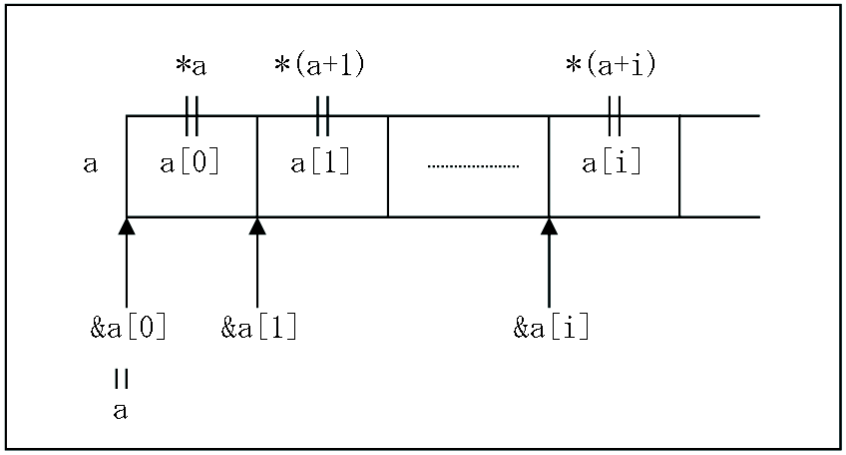
0x7ffff9ec 0x7ffff9ec

さあ、いかがですか。　a=&a[0]　という関係はしっかり覚えてくださいネ。

練習問題 2.2.2　配列 a[10]について、

*(a+1)

は何を意味するか。この事実を char 型配列を用いて確かめよ。



図（*(a+i)=a[i]）

２．３　ポインタと配列

配列名が、配列の先頭アドレスを意味することを先に述べました。この規則を適用すれば、a[i]は*(a+i)に書き換え可能でした。このことは、一般のポインタについても当てはまります。次の例題を見てください。

例題）ポインタ演算の簡便記法

```
main(){
        char    *x;     /*  x は  char 型変数へのポインタ型変数  */
        char    c[4]="abc";
        x=c;            /*  x に c [0]のアドレスを代入する */
        printf("%c %c %c¥n",x[0],x[1],x[2]);
}
```

結果

a b c

まとめ

Ｃ言語プログラムでは、以下の様な”読み替え”が行われる。

配列 a について

a=&a[0]

a[i]=*(a+i)

ポインタ p について

p=&p[0]

p[i]=*(p+i)

練習問題２.3.1

次のプログラムの実行結果を予測せよ。

```
main(){
        char    *p;
        char    c[5]="abcd";
        p=c;
        *(p+1)='x';
        printf("%s¥n",p);
}
```

練習問題 2.3.2

（１）char 型変数を３つ（a,b,c)宣言し、適当な値を代入する。

（２）char へのポインタ型変数からなる配列 d を宣言する（成分３）。

（３）d の各成分は、ａ，ｂ，ｃのアドレスとする。

（４）ａ，ｂ，ｃの値を表示する方法を３通り以上考える。

練習問題 2.3.3

以下のプログラムをポインタを用いて作成せよ。

（１）gets を用いて、char 型配列に適当な値を代入し、
この配列の内容を、先頭２文字のみ読み飛ばして表示するプログラム

（２）gets を用いて、char 型配列に適当な値を代入し、
この配列の内容を表示するプログラム。ただし、スペースは表示しない。

練習問題 2.3.4

以下のプログラムをポインタを用いて作成せよ。

（１）char 型配列（成分１０）を３つ宣言する（a,b,c とする)。

（２）gets を用いて、キーボードからそれぞれに文字列（すべて英小文字）を入力する。

（３）先頭文字のみに着目して、a,b,c の内容をアルファベット順に表示する。

練習問題 2.3.5（２分木の問題）

以下の構造体を考える。

```
struct node{
        char question[50];
        struct node * yes;
        struct node * no;
};
```

この構造体を用いて、ｙｅｓとｎｏだけで答えるゲームを作成せよ。サンプルプログラム（実行ファイル）は、~/j06/problems/yes_no_game

動作例）

以下の質問に、y（yes)かｎ(no)で答えてください
りんごは好きですか（ユーザがｙと入力）
みかんは好きですか（ユーザがｎと入力）
ビタミン不足が心配です。おわり。

２．４　メモリの動的確保

次のようなプログラムを考えます。

例題）知人の名前（アルファベット）を文字型配列に格納する。

```
main(){
        int     i;
        char    a[20];      /* 名前の長さが 20 以内とする。*/
                                /* 3 人まで格納できる  */
        char    b[20];
        char    c[20];
        gets(a);
        gets(b);
        gets(c);
        printf("%s¥n",a);
        printf("%s¥n",b);
        printf("%s¥n",c);
}
```

この例題では、格納できる人数を３人と決めておいて、人数分の配列を最初に確保しています。勿論、その人数を超えた入力は受け付けません。二次元配列を使って、

```
main(){
        int     i;
        char    a[3][20];
```

```
        for(i=0;i<=2;i++)
        gets(a[i]);
        for(i=0;i<=2;i++)
        printf("%s¥n",a[i]);
}
```

としても、人数を３人に限定していることには変わり在りません。それでは、人数を増やして、

```
main(){
        int     i;
        char    a[100][20];
        for(i=0;i<=99;i++)
        gets(a[i]);
        for(i=0;i<=99;i++)
        printf("%s¥n",a[i]);
}
```

とするとどうでしょう。これなら、当面大丈夫ですが、入力する人数が少なければ、メモリがもったいないし、100 人を越えたときは、対応できません。このような、ユーザのわがままな要求に柔軟に答えるテクニックをお教えしましょう。

例題)）知人の名前（アルファベット）を格納する。

```
main(){
        int     n,i;
        char    *p;
        scanf("%d",&n);                 /*  入力する人数ｎを入力させる  */
        getchar();                      /*  バッファに残った¥n を消す  */
        for (i=0;i<=n-1;i++){
                p=(char *)calloc(20,1);  /*
                                         １バイトの領域を 20 個確保する。
                                         calloc の戻り値は
                                         確保した領域の先頭アドレス
                                         */
                gets(p);                 /*
                                         p が示すアドレスから
                                         文字列を格納する。'¥0'を含めて
                                         ２０文字まで
                                         */
```

```
        }
}
```

上のプログラムのように、プログラム中で、必要に応じてメモリを確保する操作を、”メモリを動的に確保する”と言います。

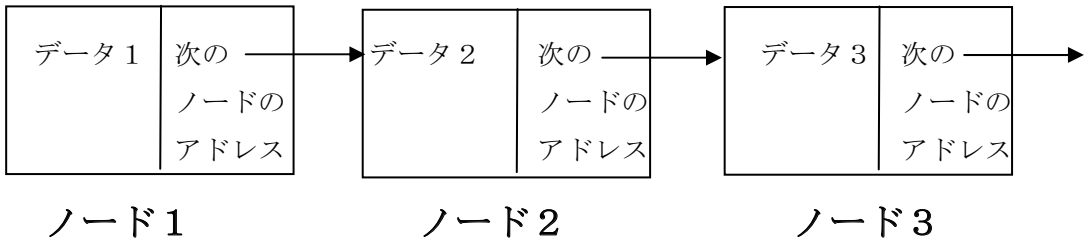
練習問題 2.4.1　メモリを動的に確保するテクニックを使って、練習問題 2.3.4 を改良せよ。改良の方針：①文字列を格納するのは配列ではなく、動的に確保したメモリ領域とする。②文字列の個数は３つとは限らない。scanf を用いて個数をあらかじめ入力する。③文字列のアドレスを格納するサイズ１００の配列をあらかじめ宣言しておく。

練習問題 2.4.2　上の練習問題 2.4.1 のプログラムは最初に入力する人数が決まっている場合は有効だが、決まっていない場合は有効でない。決まっていない場合にも使えるプログラムをつくれ（「～が入力されるまで、入力処理を繰り返す」というプログラムをつくる）。

練習問題 2.4.3（チャレンジ）　上の練習問題 2.4.2 のプログラムを改良して、入力、表示、削除が出来るプログラムを作成せよ（たとえば、ｉキーで入力、ｐキーで表示、ｄキーで削除という具合に工夫する）。

練習問題 2.4.4（リスト構造）　練習問題 2.4.2 をリスト構造を用いて実現せよ。リスト構造とは、練習問題 1.7.4 で見たような構造である。すなわち、複数のデータを格納する際に、各データの内部に次のデータへアクセスするための情報を書き込んだ構造である。

（ヒント）



実際には、以下のような構造体 struct friend を宣言し、一つのノードと考える。

```
struct friend{
        char name[20];
        struct friend * next ;
};
```

プログラムではまず、一番目のノード（２４バイト領域）を確保(　calloc(24,1);　) し、さらに、この２４バイトを struct friend として使う為に型変換する。まとめると、

```
p=(struct friend *) calloc(24,1);
```

となる。gets で名前を入力し、さらに次のデータを確保、そして、そのアドレスを格納する。

```
gets((*p).name);
(*p).next=(struct friend *) calloc(24,1);
```

これで、ノード１の作成は完了した。次に

```
p=(*p).next;
```

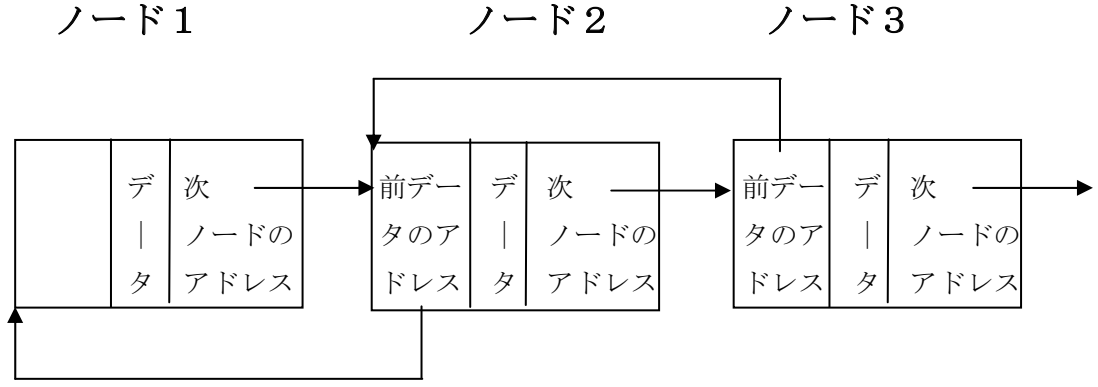
として、再び、

```
(*p).next=(struct friend *) calloc(24,1);
```

とすれば、ノード２の作成ということになる。　　（ヒント終わり）

（補足）C 言語では(*p).name の同義語として p->name という命令が用意されている。記号 ->　はアロー演算子と呼ばれる。アロー演算子を用いた方が、プログラムが見やすく間違いも少ないので慣れた方がよい。

練習問題 2.4.5（双方向リスト構造）　上の例題でリスト構造を学んだ。リスト構造をさらに発展させて、双方向リスト（両方向リストとも言う）というデータ構造を考える。双方向リストとは以下の図のように、各ノードが、前ノード及び次ノードのアドレスを同時に持っているリスト構造である（ノード１の先頭の空白は、任意のアドレスを代入して良い)。双方向リストを用いて練習問題 2.4.4 のプログラムを改良せよ。



練習問題 2.4.7（スタック）双方向リストを用いて、スタックを作成してみよう。スタックとは、複数のデータを格納する為のデータ構造である。通常スタックへのデータ格納は、PUSH と呼ばれ､データ取り出しは POP と呼ばれる(アセンブラでもおなじみであろう)。関数 PUSH と POP を作成せよ。ヒントとして、配列でスタックを構成しておくので参考にして欲しい。

```
#include <stdio.h>
int    STACK[50],stack_pointer=0;

void PUSH(int N){
        STACK[stack_pointer]=N;
        stack_pointer++;
}
// put the number that is in the STACK, to the valuable N

int POP(void){
        stack_pointer--;
        return STACK[stack_pointer];
}
// 0 が入力されるまでスタックに積む（格納する）。積み終わると pop して表示。
main(){
int N=1;
        while(N!=0){
                N=getchar()-'0';
                getchar();
                PUSH(N);
                }
        while(stack_pointer!=0)
                printf("%d¥n",POP());
}
```

補足：リストを用いてスタックを実現するプログラムの中では、以下が最も簡単であろう。

```
struct node {
        int data;
        struct node *next;
} ;
struct node *root=NULL;

void PUSH(int n){
        struct node *tmp;
        tmp=(struct node *)calloc(sizeof(struct node),1);
        tmp->next=root;
        tmp->data=n;
```

```
        root=tmp;
}

int POP(void){
        int tmp=root->data;
        root=root->next;
        return tmp;
}
```

練習問題 2.4.8（キュー）キューというデータ構造について、教科書を読み、プログラムを作成せよ。実現手段はさまざまである。　配列で構成するか、ポインタを用いて構成するか。静的に構成するか、動的に構成するか等、さまざまな選択肢がある。各自の実力に応じて構成して良い。