プログラミング入門 II

動的なメモリ割り当て と リスト構造

これまでのおさらい（ポインタと構造体）

- ポインタ
  - ポインターってなに？
  - ポインター変数
  - 関数の引数とポインター
  - 配列とポインターの関係
  - データ型
- 構造体
  - 構造体ってなに？
  - データ型と構造体
  - Typedef
  - ポインターを使った構造体の参照
- 動的なメモリ割り当て

リスト構造
＝
構造体
＋
ポインタ
＋
動的な
メモリ割り当て

ポインターとは？

- ポインタを使うと*効率的なプログラムを、わかりやすく書くこと*
  - Ｃ言語の中で、最も便利で強力な仕組み
  - ポインタはなくても、プログラムは書くことができますが、…
- ポインタは計算機の仕組みと密接に関係しており、計算機の仕組みをわかりやすくプログラマに見せてくれる
  - Ｃ言語がオペレーティングシステムなど計算機のハードウエアに近いところを扱わなくてはならないシステムプログラムによく使われる理由

ポインタと計算機の仕組み

- ポインタとはアドレスのこと！
- アドレスとは？
  - メモリは１バイト（８ビット）ごとに区切られており、ＣＰＵがメモリにアクセスする場合には、何番目のバイトかを指定してアクセスします
  - この何番目かのメモリかがアドレス
  - 数を格納するためには整数は３２ビットつまり、４バイトが必要なので、４つの連続したバイトを使って格納しています

ポインターとアドレス

- メモリとはデータをいれておくための箱
- アドレスとはその箱につけられている番号
- ポインタとはこのようなアドレスをあらわす値の、Ｃ言語での呼び名

アドレスを得るには？

- 変数名に&をつけると、変数のアドレス、つまり、*変数へのポインタ*になります

```
int x;
scanf("%d",&x);
```

- アドレスをscanfに渡す！

ポインター変数

- アドレス（ポインタ）を格納するための変数をポインタ変数といいます。

```
int *p;
```

- C言語では、ポインタ変数の宣言にはそのポインタがどのようなデータ型を格納しているかを指定しなくてはなりません。
- 変数の宣言のデータ型として、値のアドレスのところに格納するデータ型とその後に*をつけて、宣言します

ポインタ変数の使い方

- 変数xへのポインタをポインタ変数pに格納する

```
p = &x;
```

- ポインタ変数に格納されているアドレスのところにある整数を読み出す

```
y = *p;
```

```
y = *p + 100;
```

- * は、ポインタ変数pで指されているデータを参照する

ポインター変数の使い方

- *pは、代入に使うこともできます。

```
*p = 100;
```

- pのさしているところに、１００をいれる

関数とポインタ

- scanfでは、&xとして整数変数のxのアドレス、つまり「xへのポインタ」を引数にしていました。
- ポインターを引数とする関数を定義するには、関数のパラメータとしてポインタ変数を宣言する
- 例えば、２つの変数の値を取り替える関数swapは以下のように定義できます。

```
void swap(int *p, int *q){
    int t; t = *p; *p = *q; *q = t;
}
```


１つ以上の値を返したい時

- これまで説明した関数は関数の返り値として、return文で１つの値しか返すことができませんでした。
- ポインタの引数を使えば、複数の値を返すことができます。例えば、足し算と引き算の値を同時に返す手続きは以下のようにします。

```
void addsub(int x, int y,
            int *add, int *sub){
   *add = x + y;
   *sub = x - y;
   return;
}
```


配列とポインタ

- 配列Aとは、１００個の整数分の連続したメモリを確保して、それにAと名前をつけたもの

```
int A[100];
```

- Aという名前はその*配列のメモリのアドレス*そのものを表します

```
int *p;
 p = A;
```

- ポインタ変数に代入できる

データ型とは

- データ型: 変数や配列がどのような種類の値をもっているか
- 基本データ型
  - int
  - float
  - char
  - ....

データ型　変数名、変数名、…;

データ型　配列名[配列サイズ][…];

データ型とは

- ポインターもデータ型

データ型　＊

- データ型をさすポインタ型

```
int *ip;
char *cp;
```


構造体（structure)とは

- 構造体とは、プログラムを論理的にわかりやすく書くための機能です。
  - プログラムとは、人間のプログラマにとってはコンピュータの中で何かをさせたい場合にそれを表現するための言語。
- 構造体とは、いくつかの要素を持つデータを表現するためのデータ型

成績表の例

- たとえば、成績表を集計するプログラムを考えてみましょう。
  - 算数、国語、理科と３科目とすると、人数を縦、３科目の点数を横にとる表をつくります。
  - これをプログラムで表現するには２次元配列をつくります。

```
int seiseki_hyou[50][3];
```

例えば、１０番の人の国語の成績を参照するには

```
seiseki_hyou[9][1]
```

となる

- わかりやすいか？

構造体を使うと

- 算数と国語、理科の成績をひとまとまりにしてデータ（の型）として名前をつけておくことができます。

```
struct seiseki {
  int sansu;
  int kokugo;
  int rika;
};
```

- これをつかって、表にする

```
struct seiseki seiseki_kyou[50];
```


構造体データ型の宣言

- 宣言

```
struct 構造体の名前 {
    フィールドのデータ型 フィールド名;
    フィールドのデータ型 フィールド名;
  …
};
```

- 構造体の要素のデータをメンバー、あるいはフィールドという。いろいろなデータ型を使える。

```
struct personal_record {
   char name[10];      /* 名前をいれる */
   int age:            /* 年齢 */
   char address[100]; /* 住所 */
   int tel[11];        /* 電話番号 */
};
```

構造体の変数、参照

- 構造体のデータ型を持つ変数の宣言

  struct 構造体の名前　変数名;

- 参照のメンバーの参照

  構造体への参照 . メンバー名

構造体の変数、参照

- ２次元座標上の点をあらわすには次のような構造体を宣言します

```
struct point {
    double x, y;
};
```

- 点のデータ型を持つ変数の宣言

```
struct point p;
```

- X座標のデータの参照

```
t = p.x;
p.x = 199
```

```
struct point {
   double x,y;
} p;
```


構造体の配列

- 構造体の配列の宣言

  struct 構造体の名前　配列名[サイズ];

  - １０個の点の配列

    struct point points[10];

- 参照
  - ５番目の点のｘ座標

    points[４].x

  - ３番目の人の国語の成績

    seiseki_hyou[２].kokugo

データ型とは（再び）

- データ型をTとすると
  - 変数の宣言

    T　変数名；
  - 配列の宣言

    T　配列名[サイズ];

- 構造体のところでは、

  Tが struct 構造体名

  になっていることに注意。

- 構造体はデータ型である！

typedef宣言

- データ型に自分の名前を付けることができる。

  typedef T データ型名;

```
typedef struct point {
    double x,y;
} point_t;

point_t p;
point_t points[10];

typedef char * string_t;
```

あるデータ型に対して、適当な名前をつけておけば、プログラムがわかりやすくなり、書きやすくなります。

プログラミングとは、やりたいことを論理的にわかりやすく表現することでもあるのです。

構造体の代入

- 同じデータ型同士であれば、代入ができる。
  - コピーする
- 演算はできない！

```
struct point {
   int x,y;
};     /* 構造体の定義 */
struct point A,B;
       /* 構造体変数の定義 */
…
A = B;  /* 代入、コピー */
…
```

構造体の引数・返り値

- 引数にも使える
  - ただし、コピー
- 値渡し(Call by Value)
- 返り値にも使える
  - これもコピー

```
struct point goo(…){
   struct point X;
   ….
   retrun X;
}
```

```
struct point A,B;

void foo(struct point a,
         struct point b){
   …
}

   …
   foo(A,B); /* 引数 */
   …
   A = goo(); /* 返り値 */
```


構造体のサイズ

- 代入、引数、返り値
  - どれもコピーされる！
- では、どのくらいのデータがコピーされるのか？
- sizeof ：サイズ（バイト単位）を調べる演算子

```
printf("size is %d¥n",sizeof(struct point));
```

いくつとプリントアウトされるか？

構造値とポインタ

- 構造体を上のように、直接、引数に使ってしまうと、コピーされるため、大きな構造体の場合、不効率になってしまうことがある

- そこで、ポインターをつかって引き渡す
  - コピーされるのはアドレス（4バイト）だけ
  - ポインタは効率的なプログラムを書くためのしかけ

```
void foo(struct point *ap,
         struct point *bp){ … }
…
foo(&A,&B);  /* ポインタを渡す */
```


ポインタを使った構造体の参照

- ポインタからメンバーの値を参照するには、*->演算子*をつかいます

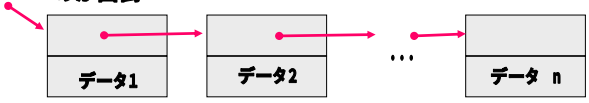
```
struct point *ap;
   …
t = ap->x + 1; /* メンバーxを参照*/
ap->x = 123;   /* メンバーxへ代入*/
```

`ap->x ⇔ (*ap).x`

（線形）リスト構造とは

- 一列に並んだデータをあらわすデータ構造
  - 集合、順序がついたものの並び
  - 必要に応じて、データを確保

- 配列だと
  - データの数の上限をあらかじめ決めておかなくてはならない。
  - 途中にデータを挿入したり、途中のデータを削除するのが面倒



リスト構造の定義

- 自分を参照するポインタのメンバーを持つ

```
struct List {
   struct List *next;
   int data;
};
```

ここのデータは場合によっていろいろなものをつけることができる！

実行中にメモリが欲しくなったら…

- メモリを確保する関数
  malloc(確保するバイト数)
- データ型を強制的に指定する演算子：キャスト

```
ap=(struct point *)malloc(sizeof(struct point));
```

キャスト演算子 データ型を指定

malloc関数でメモリを確保

sizeof演算子で欲しいメモリサイズの計算

メモリを「動的に」わりあてる

リストへのデータの追加

- 大域変数として、リストを持っている変数head
- データを追加する関数addList

```
struct List *head = NULL;

void addList(int x){
  struct List *lp;
  lp = (struct List *)malloc(sizeof(struct List));
  if(lp == NULL){
     printf("no more memory¥n");
     exit(1);
  }
  lp->next = head;
  lp->data = x;
  head = lp;
}
```


リストにあるデータの検索

- リストにデータがあったら、１、なかったら０を返す関数 isExist

```
void isExist(int x){
  struct List *lp;
  for(lp = head; lp != NULL; lp = lp->next){
    if(lp->data == x) return 1;
  }
  return 0;
}
```


メモリの開放

- メモリを開放するにはfree関数を使う

```
free(mallocでもらったpointer);
```

- 開放されたメモリは、後でmallocでつかわれる
- 注意：malloc, freeを使うには、stdlib.hをincludeしておく必要があるので忘れずに。

リストにあるデータの削除

- リストにデータがあったら、削除する removeList

```
void removeList(int x){
  struct List *lp,*lq;
  lq = NULL;
  for(lp = head; lp!= NULL; lp = lp->next){
    if(lp->data == x){
       if(lq == NULL) head = lp->next;
       else lq->next = lp->next;
       free(lp);
       return;
    }
    lq = lp;
  }
}
```


練習問題

- データが順に並んでいるとして、途中にデータを挿入する方法を考えなさい