フローチャート図を利用したプログラムの設計法

はじめに

皆さんはコードを書いている時、何度も書き直したり、書き直す際にあちこち書き直す必要に迫られたことはありませんか。もしくは、細かいアルゴリズムを要する機能を実装する時、頭が混乱したことはありませんか。もしくは、作りたいプログラムがあるけど機能が複雑でどう作れば良いのかわからないという経験はありませんか。これらは事前に設計をせず、すぐにコードを書き始めてしまったことが原因であることが多いです。そこで、この講座資料ではフローチャート図を利用して、プログラムの流れを視覚化し、設計する方法をご紹介します。

プログラムの構築法の歴史

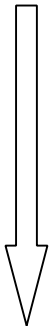
設計の方法を学ぶ前に、まず今回の設計の仕方はプログラミングパラダイムのどのあたりに位置するものなのかを把握しておくと後々のスキルアップに役に立つでしょう。そこで、まずはプログラムの構築法の歴史について述べていきたいと思います。

あらかじめお断りさせてもらうと、ここで述べるプログラムの構築法の歴史は筆者が便宜上定義したものであって、一部正しくないところも存在します。本当にしっかりした歴史を学びたい場合は wikipedia を参照ください。また、ソースコードは C++で記述してあります。

原始的なプログラム

まずは例を見てみましょう。

```
#include<stdio.h>
int main()
{
        int a=0;
        int b=0;
        scanf("%d",&a);
        if(a==1)
        {...}
        ...
}
```

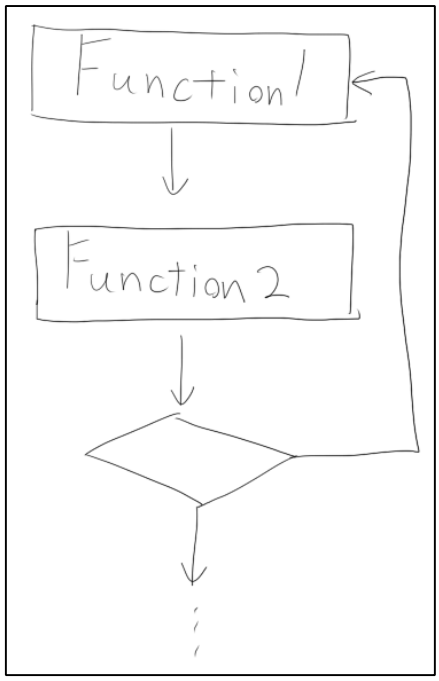


上から下にただ
最小単位の命令
を並べるだけ！

以上のように、最小単位の命令を上からそのまま書いていくだけのこのプログラムの構築法を筆者は便宜的に「原始的なプログラム」と呼んでいます。ここで使用される命令は、「int a」や「if」など、最小単位の命令です。非常に細かい命令の集合で成り立っているのがこの原始的なプログラムです。原始的なプログラムは初心者にとって書きやすいですが、少しでも複雑なプログラムになるとすぐスパゲッティコードになります。そのため、少しでも複雑なものを作ろうとするならばこの作り方はしないほうが良いでしょう。

手続き型プログラム

```
#include<stdio.h>
void func1(int a,int b)
{...}
int func2(int a,int b)
{...}
int main()
{
        int a = 0;
        int b = 0;
        func1(a,b);
        func2(a,b);
        return 0;
}
```

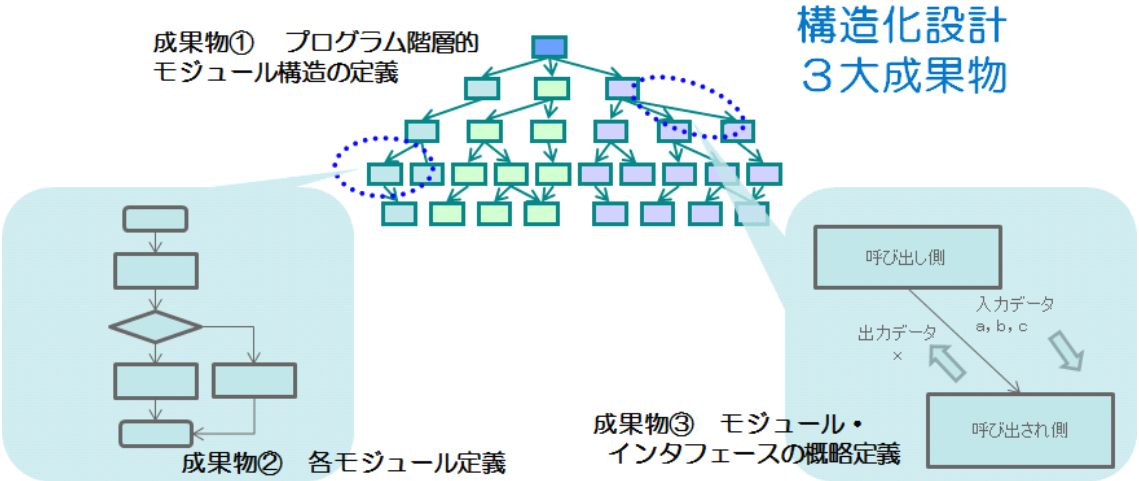


手続き型プログラムの概念図

以上のように、関数で機能を分割して記述するプログラム構築法を筆者は便宜的に「手続き型プログラム」と呼んでいます。今回の講座資料で使うプログラム構築法はこの手続き型プログラムを用います。手続き型プログラムは、ちょっとした処理だけをしたいツールを作るのに便利ですが、複雑なプログラムになると不向きです。

構造化プログラム

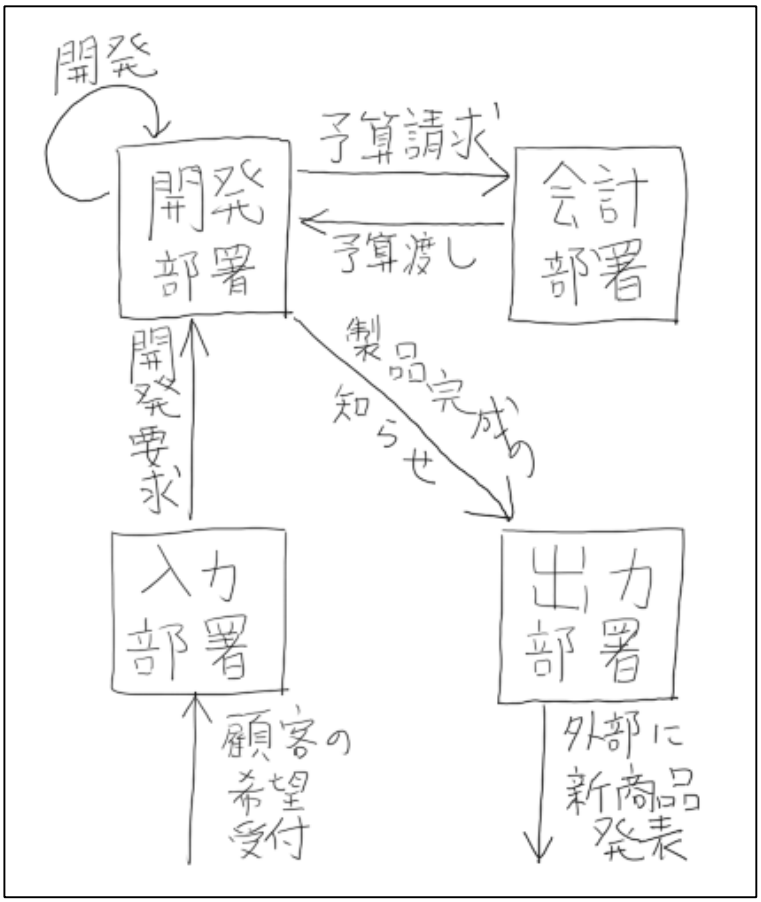
手続き型プログラムにモジュール性を導入したプログラム構築法。今回の内容とは関係ない上、手続き型とかぶっている部分が多いため、概念図だけ載せてあとは割愛します。なお、構造化プログラムは複雑である程度巨大なプログラムを作るのに向いています。



オブジェクト指向プログラム

今回の講座資料では用いませんが、この先利用することになると思うので、オブジェクト指向のプログラムも見てみましょう。

```
class Hoge
{
private:
        int a;
        int b;
public:
        Hoge();
        ~Hoge();
        void func(int in1,int in2);
};
int main()
{
        Hoge* exam = new Hoge();
        exam->func(3,5);
        delete exam;
}
```

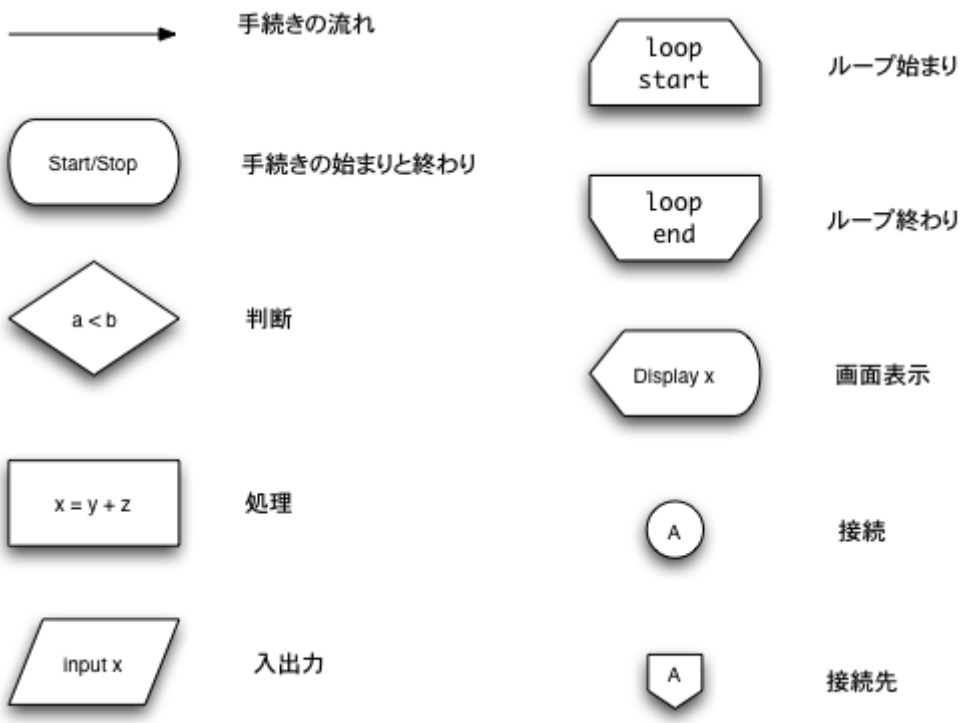


オブジェクト指向プログラムの概念図(製品開発の例)

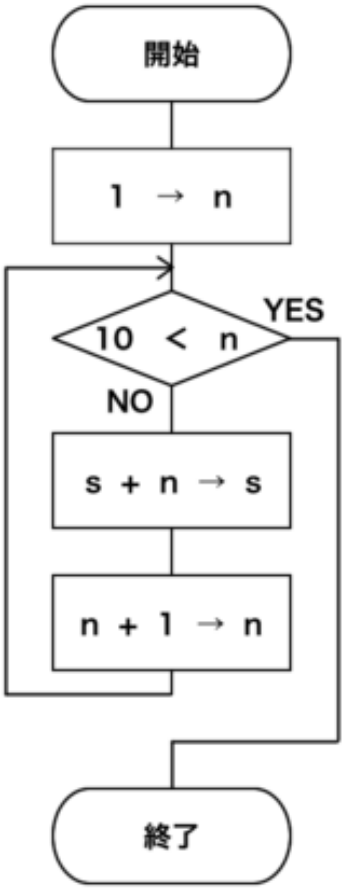
このように、クラスというものを実体化したインスタンスを用いてプログラムを構成する方法です。このインスタンスとは、人間世界で言う「部署」だとか、「局」といった働きをし、1 インスタンスにつき機能は 1 つです。つまり、プログラム全体を構成する数ある機能をそれぞれ分担した形と言えます。オブジェクト指向は、非常に巨大で複雑なプログラムを作成するのに非常に便利な考え方です。ゲームを作成する際もオブジェクト指向でプログラムを構成するのが良いでしょう。

フローチャート図の紹介

今回使用するプログラム構成法は手続き型です。この構成法を用いてプログラムを設計するのですが、設計するのに必要なツールを紹介します。それがフローチャート図です。



そして、上のフローチャート記号を実際に使用した例が以下の図です。



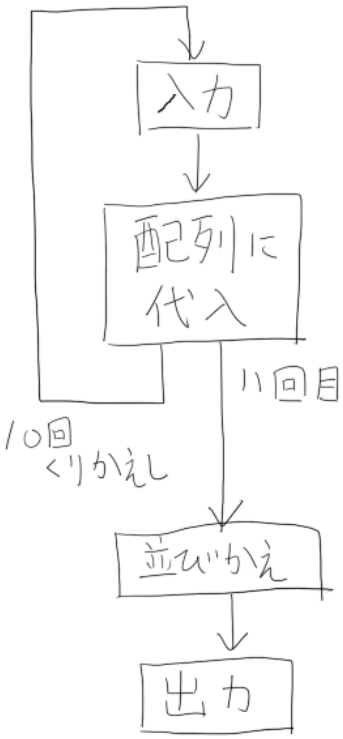
この図は「1～10 までの自然数の和」を計算するプログラムを表しています。開始後に「n=1」で初期化し、「10<n」になるまでひたすら値を足していってます。

以上の例を見てみると、「こんなきっちり書かなければならないのだろうか」とか、「ここまで細かい処理を書かないといけないのだろうか」とか思う方がいるかもしれませんが、そんなことはありません。たしかに Web 上で適当に拾ってきた画像なので例はきっちりと書かれているものを載せましたし、処理内容も「n=1」だとか「n=n+1」だとか最小単位の命令ばかりです。ですが、こんなきっちり書かなくても自分が理解できれば良いですし、なによりフローチャート図の真価は「より巨視的に見た処理をも表せる」というところにあるのです。

フローチャート利用例

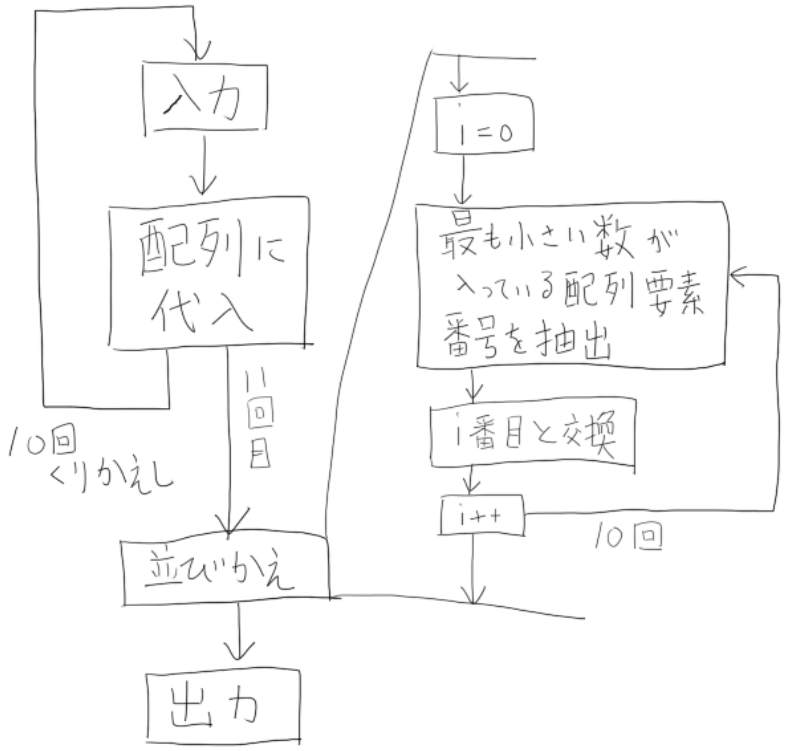
例えば「数値の入力を 10 回受け付けた後、その 10 個の数値を小さい順に並び替えて出力せよ」という問題を考えてみましょう。

この問題を解く際に筆者は以下のようなメモ書きをしました。



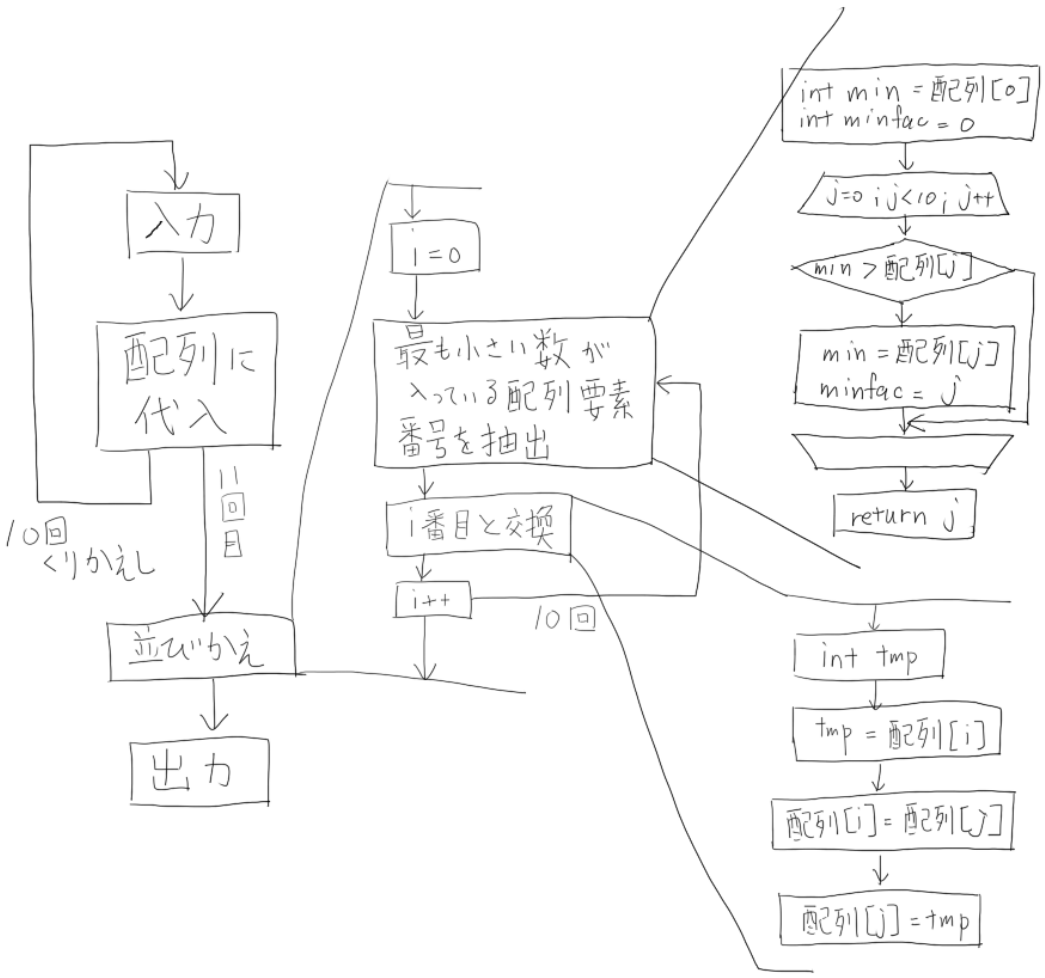
このメモ書きを見る限り、しっかりとした書き方になんて全く則ってないですし、書いてある処理も超大雑把です。しかも手書きです。フローチャートの良いところはこんな書き方をしても処理の流れがしっかりと見えるという点です。要は自分が理解して流れを整理できれば良いのです。

しかし、繰り返し部分はこれでソースコードに落とし込むことができるかもしれませんが、並び替えのあたりがまだソースコードに落とし込むことができません。ソースコードに落とし込むにはまだ大雑把すぎるのです。そこで、並び替えの処理だけ注目してさらに処理を細分化します。



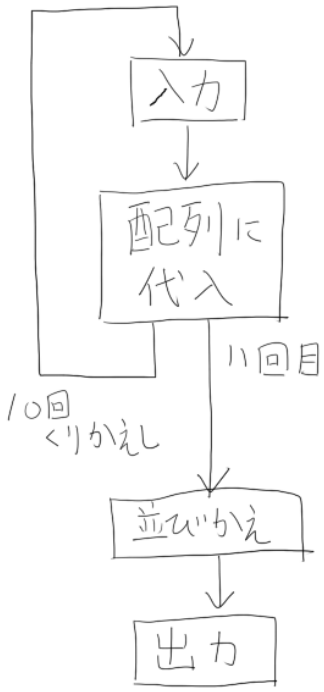
並び替えのところだけさらに細分化させたメモ書きが以上です。

だんだんと最小単位の命令の集合体としてのプログラムが見えてきました。もし頭の回転の速い人ならここまできたらソースコードに落とし込むことができるかもしれません。しかし筆者は決して頭の回転が速いとは言いがたいので、まだソースコードに落とし込む自信がありません。特に「最も小さい値の入っている配列要素を抽出」と「i 番目と交換」という処理がいまいちコードに落とす自信がありません。そこで、この 2 つの処理をさらに細分化します。



細分化した結果、以上のようなフローチャートができあがりました。ここまで来たらどうやらコードに落とし込むことができそうです。早速実装してみましょう。

まず最初に書いた図を思い出してください。



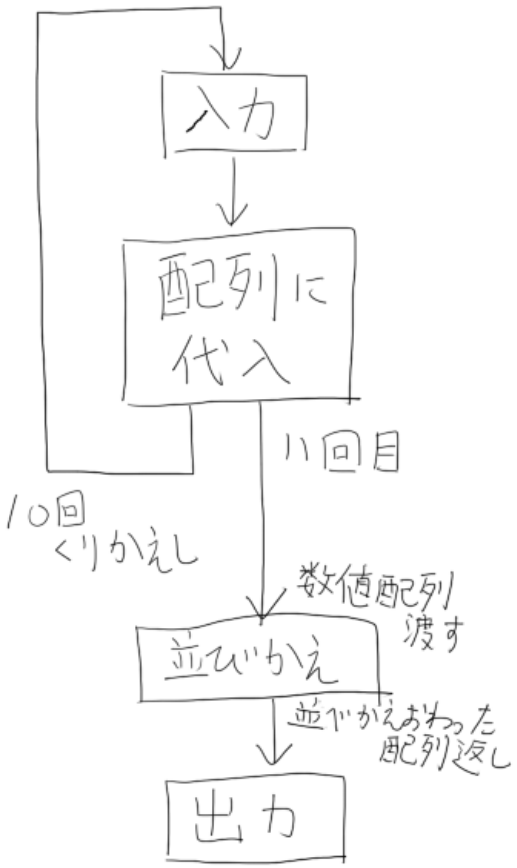
まずはこの図だけをソースコードにしてみましょう。大雑把な処理として書いた部分（今回の例だと並び替え処理の所）は関数を用います。

```
#include<stdio.h>
void sort(int num[])       //並び替え関数定義
{
        //とりあえずは空にしておく
}
int main()
{
        int num[10];       //入力した値を格納する配列
        puts("数値を10回入力してください");
        for(int i=0;i<10;i++)      //入力繰り返し処理
        {
                scanf("%d",&(num[i]));
        }
        sort(num);         //並び替え関数
        for(int i=0;i<10;i++)      //出力
        {
                printf("%d¥n",num[i]);
        }
        return 0;
}
```

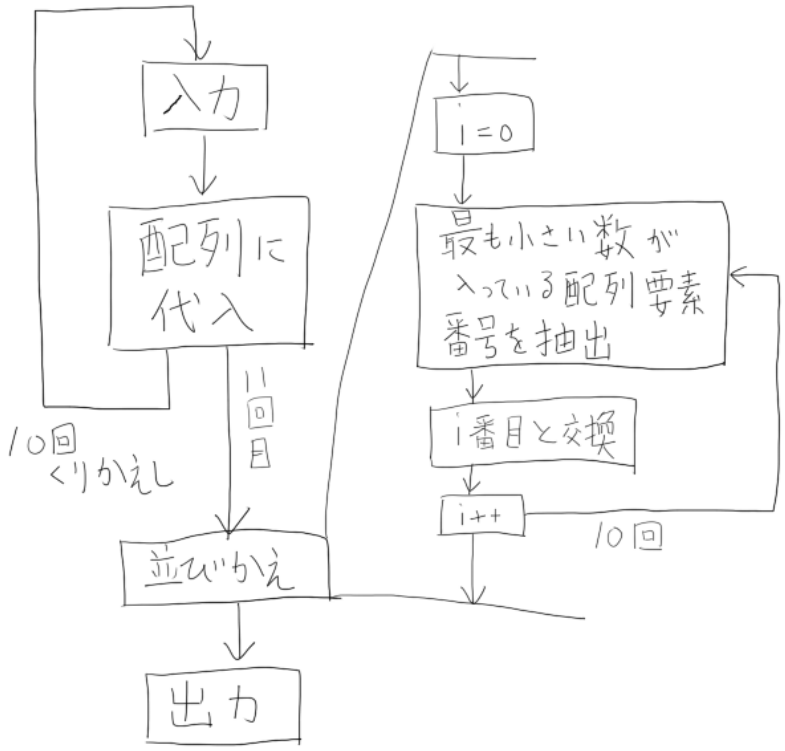
実装している途中で、フローチャート図のほうでは書かれていない問題にぶつかります。ずばり、「並び替え関数にはどんなデータを渡し、どんなデータが返ってくれば良いのか」ということです。これを解決するために、フローチャート図を書いた時点でこの「並び替え処理」がどのような処理をしてくれるものだと期待していたのかを思い出す必要があります。すると、並び替えをする前の配列を渡し、返し値は並び替え終わった配列であればよいことがわかります。そこで、sort 関数では引数に配列を入れ、配列要素番号を返すように記述しておきました。

返し値は関数内の最後で return によって返しても良いのですが、今回は引数にポインタ（配列を引数に入れるということは配列の戦闘アドレスを格納したポインタを渡すことと同値）を入れることでポインタ越しに関数内で配列を編集するようにしました。

今加えたデータの流れを加えて先ほどのフローチャート図を更新しておきましょう。



では次に中身を空にしている sort 関数を埋めましょう。2 枚目の図を思い出してください。



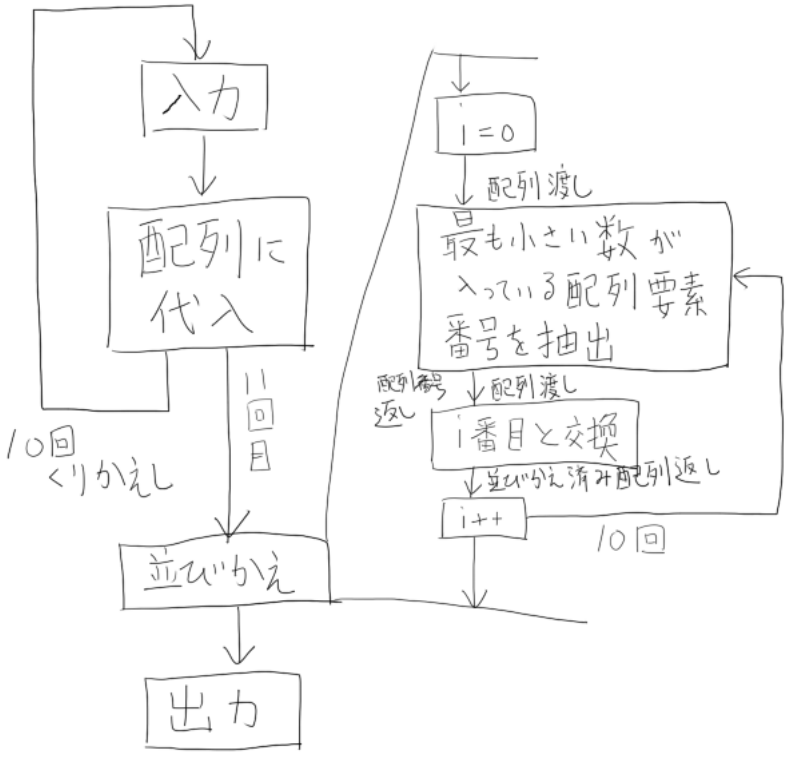
このさらに細分化した処理を sort 関数の中に書いていけば良いわけです。

```
int getMinFac(int num[])
{
        //まだ空のまま
}
void exchange(int num[],int from,int to)
{
        //まだ空のまま
}
void sort(int num[])       //並び替え関数定義
{
        int minfac;
        for(int i=0;i<10;i++)
        {
                minfac = getMinFac(num);   //最小の数が入っている要素を抽出
                exchange(num,i,minfac);              //要素を交換
        }
}
```

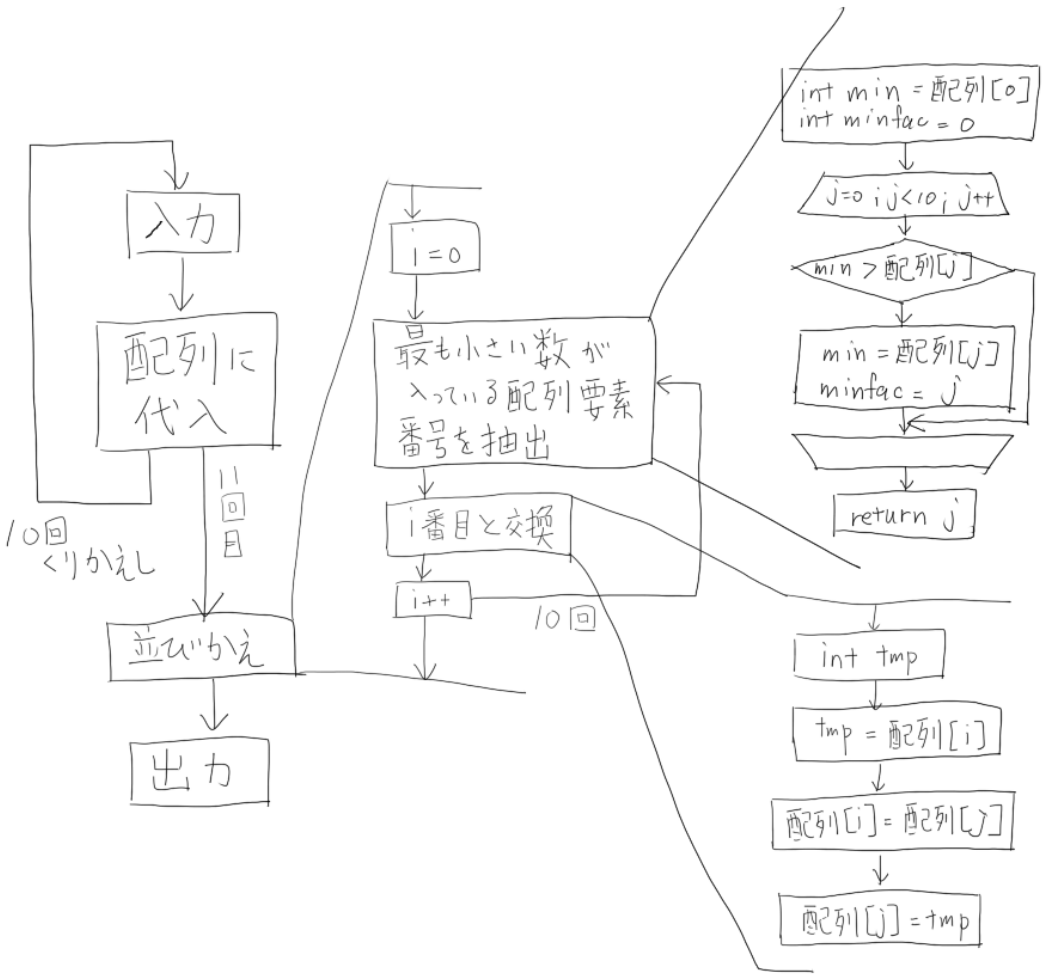
sort 関数の中で必要になったので getMinFac 関数と exchange 関数の空定義だけ記述しておきました。それぞれ「最小の値が入っている配列要素を返す関数」「配列の要素を交換する関数」です。

getMinFac 関数では配列要素を渡して最小の値が入っている配列要素番号を返し値で返すようにしました。また、exchange 関数では配列と移動前要素番号と移動後要素番号を渡し、引数のポインタごしに返すように設定しました。

データの流れを含めて 2 枚目を更新しておきましょう。



そして 3 枚目についても同様にコードにしていきます。


入力
配列に代入
11回目
10回くりかえし
並びかえ
出力
i=0
最も小さい数が入っている配列要素番号を抽出
i番目と交換
i++
10回
int min = 配列[0]
int minfac = 0
j=0 ; j<10 ; j++
min > 配列[j]
min = 配列[j]
minfac = j
return j
int tmp
tmp = 配列[i]
配列[i] = 配列[j]
配列[j] = tmp

```
int getMinFac(int num[])
{
	int min = num[0];
	int minfac = 0;
	for(int i=0;i<10;i++)
	{
		if(min>num[i])
		{
			min = num[i];
			minfac = i;
		}
	}
	return minfac;
}
void exchange(int num[],int from,int to)
{
	int tmp = num[to];
	num[to] = num[from];
	num[from] = tmp;
}
```

書き終わったら実行してみましょう。すると、実行してもうまく並び替えができないことがわかります。つまり、このコードのどこかにバグがあるということになります。

フローチャートを利用した設計の怖いところは、設計者が大雑把に設計しすぎてどこか必要なところを見落としている可能性がある点です。そのため、設計して実装し終わったら見落とした点がないか、しっかりとデバッグしてあげる必要があります。

ずばり言うと、今回の例では最小の値が入っている配列要素を返す処理のところ（getMinFac 関数の中）に誤りがあります。exchange 関数によって配列の中身が入れ替えられた後でもこの getMinFac 関数では 0 要素目から 9 要素目まで全部調べてその要素番号を返しています。最も小さい値が num[0]に入れ替えられた後は毎回返す要素番号が 0 になってしまうため、入れ替えられないという事態に陥っています。

そこで、getMinFac 関数を以下のように変更します。

```
//引数を1つ追加。cont番目の要素以降のみから最小の値が入っている要素番号を抽出
int getMinFac(int num[],int cont)
{
	int min = num[cont];		//最小の値をcont番目の要素の中身に変更
	int minfac = cont;		//同様に暫定的にcont番目の要素番号に設定
	for(int i=cont;i<10;i++)	//cont番目以降のみ抽出対象
	{
		if(min>num[i])
		{
			min = num[i];
			minfac = i;
		}
	}
	return minfac;
}
```

そして、sort 関数内に記述されている getMinFac 関数の引数を以下のように変更します。

```
void sort(int num[])
{
	int minfac = 0;
	for(int i=0;i<10;i++)
	{
		minfac = getMinFac(num,i);//追加した引数にiを入れる
		exchange(num,i,minfac);
	}
}
```

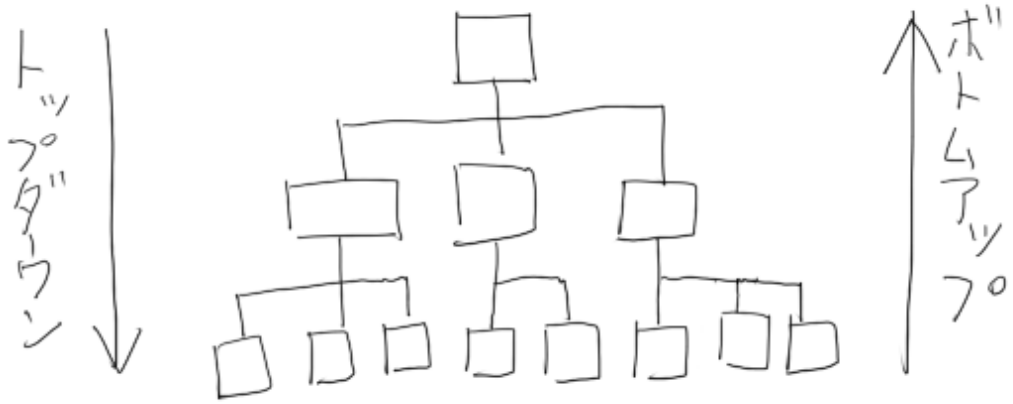
これで実行してみましょう。これで上手く動くはずです。

まとめ

ということで、今回はフローチャート図を利用した設計の仕方を説明しました。ここで

設計についての基本的な考えを述べておきます。

　プログラムを設計する時はトップダウン形式を基本とします。トップダウン形式とは、まず大雑把な流れから確定した後機能を細分化していく、という手法です（これとは逆に細かい機能・部品から全体を構成するボトムアップ形式というものもあります)。トップダウン形式の良いところは、「機能を細分化していくおかげでコードを書く時一つの機能だけに集中できる」「複雑なプログラムを実現しやすい」という点です。



フローチャートを利用した設計法の説明は以上になります。ちょっとしたツールを作りたい時はこの方法を試してみてはいかがでしょうか。