章前問題

1."wish you happiness"と表示するプログラムをキーボードを見ずに作れ
2.第 n 項までのフィボナッチ数列を表示するプログラムを作れ
3.「引数」この言葉の読み方を答えよ

第一章　ポインタの前に

ゼミ A で説明していなかったことを説明します。ここらは簡単なので適当に流していいかと思いますが、確実に理解しておかないと後で面倒なことになるかもしれません。

第一節　算術型

C 言語では一つひとつの変数は宣言時に型というものを一緒に宣言しなければなりません。int a や double ave のようにです。その宣言した型にはそれぞれ役割があり、その役割でしかその変数は使うことができません。

型名	役割	例
int	整数	0, 1, 2, 0, 123456 ...etc
float	単精度小数点	0.1, 123.456, -0.00001, 10, ...etc
double	倍精度小数点	0.1, 123.456, -0.0000000001, 10, ...etc
char	文字	a, b, #, ″, ...etc

主な型は上記の表の通りです。float 型と double 型が似たような型ですが、double は倍精度と書いてあるので、float の 2 倍の精度[ii]で数を表すことができますが、メモリも float の 2 倍消費[iii]することになります。また char 型に関しては次々章で色々説明します。

第二節　sizeof 演算子と各型の表現範囲

sizeof 演算子は若干マイナー気味な演算子ですが、何かと使えるので書きます。例えば int 型はいくつまでの数字が表すことができるのか？　とか疑問に思うことが多々あります。普通は limits.h というヘッダファイルの中に int 型の最大値や最小値が#define で定義されています。ですが、そこを見るのは面倒なので sizeof という演算子を使うと何かと便利です。

```
#inlucde<stdio.h>
int main( void){
    printf( "int 型は  %d byte 使っています¥n", sizeof( int));
    printf( "double 型は  %d byte 使っています¥n", sizeof( double));
```

---

i　脚注の権限は私がもらった

ii　double や float は指数を使って値を保持しているため、小数点以下かなりの桁まで保持できる

iii　今の PC は平気でメイン RAM が 1GB とかあるのでいくら消費しても消費したことにはならない。double 型を 100 万個宣言してたったの 8MB、他のところを改良したほうがいいプログラムになることが多いです

```
    return 0;
}
```

上記のプログラムは sizeof 演算子を使った簡単な例です。これを実行すると今の処理系[i]で int 型は何バイトか、double は何バイト使っているかがわかります。多分、int 型は 4byte、double 型は 8byte だと思います。int 型が 4byte ということは、1byte = 8bit なので計 32bit、つまり 2^{32} 個の数字を表すことができます。int と宣言したときは正負[ii]があるので int 型は-2^{31}~2^{31} くらい[iii]の数まで扱えることがわかります。

さて、数の上限はわかりましたが、もし間違ってリミッターを超えたらどうなるでしょうか？

例えば下記のプログラムを見てください。

```
#include<stdio.h>
int main( void){
    int i;
    int a = (2147483647 - 5);
    //singed long intの最大値からマイナス5の値
    for( i = 0; i < 10; i++){
        printf( "%dループ目 a = %d¥n", i, a);
        a++;
    }
    return 0;
}
```

だいたい、5 ループくらいしたらよくないことが起こることが目に見えています。どうなるかは自分で確かめてください(別にハングとかはしないから大丈夫)。

話は前後しますが、sizeof の話に戻ります。sizeof 演算子は型の大きさを調べるだけではなく、配列の大きさも調べることができます。例えば int a[10]と宣言した変数に対して sizeof(a)と書くと、配列 a は何バイト消費しているかわかります。どんなメリットがあるかはだいたい察しが付くかと思います。

第三節　章末問題

1. char 型は何バイト消費するか調べよ
2. long long int 型という long int 型よりも大きな数が扱える型がある。その上限値を調べよ(なお BCC は未対応みたい)
3. "MIKO GNYO/Linux"という 15 文字の文字列が何バイト消費するか調べよ
4. "巫女ぐにょ・りぬくす"という 10 文字の文字列が何バイト消費するか調べよ

---

i　コンパイラの種類とか、コンピュータの種類とかそんな意味

ii　実は、プラスもマイナスも扱える signed 型、正の数のみの unsigned 型という区分がある。単に int と宣言したときは singed int 型となり正負が扱える int 型になる。unsigned 型は正の数しか扱えない代わりに singed 型の 2 倍の数値まで扱える

iii　-2,147,483,648 ~ 2,147,483,647 まで。ゼロが含まれる分、2^{31} - 1 の数が上限となる

第二章　ポインタ

ここではポインタの復習をしたいと思います。ゼミ A のときに疎かにしてしまった人もきっと多いと思うはずです。今後、文字列の扱いを学ぶ上ではポインタの知識は必須ですので、ここではポインタをもう一度理解してもらいます。

第一節　間接演算子

大層なサブタイトルが付いていますが、アスタリスク「*」のことを C 言語では間接演算子と呼ぶそうです。まずは関数を跨いだ変数の中身の変更のプログラムです。

```
#include<stdio.h>
void swap( int *x, int *y){
    int buf;
    buf = *x;
    *x = *y;
    *y = buf;
}
int main( void){
    int a, b;
    scanf( "%d", &a);
    scanf( "%d", &b);

    printf( "pre : a = %d, b = %d¥n", a, b);
    swap( &a, &b);
    printf( "post : a = %d, b = %d¥n", a, b);
    return 0;
}
```

ありがちすぎて困る[i]ようなサンプルプログラムです。説明の前に用語のおさらいをします。

ポインタ……指すもの。矢印。あれ。

アドレス……場所。位置。在処。

大雑把すぎる意味ですが、これがなかなか難しい。main 関数で swap 関数を呼び出していますが、ここで引数としている&a, &b はどちらもアドレスです。普通の変数に&を付けたら、アドレスを返すということです。一方、swap 関数の中身のほう、仮引数[ii]としてとっている*x, *y はポインタです。この*x, *y は main 関数の a と b を指していますよ、ということになっています。

また、ポインタ上で元の変数のアドレスが欲しいときは、この場合 swap 関数で x とだけ書けばいいです。つまり

```
void swap( int *x, int *y){
```

---

i　これ以外にいいサンプルがないとも言う

ii　いつも思うのだが、仮引数で int*　x とやったら、文法上はポインタ x に値を代入するとなる気がするのは気のせいなのか？

```
    scanf( "%d", x);
    scanf( "%d", y);
}
```

といった感じ[i]になります。

以上のことをまとめると以下のようになります。

	値が見たい	アドレスが見たい
変数	そのまま	&をつける
ポインタ	*を付ける	そのまま

またポインタの宣言の仕方ですが、

```
int *x;
int* y;
```

このように、どちらの方法でも大丈夫です。意味の違いはありませんが、後者のほうは int ポインタ型の y という意味合いが強い気がします。

第二節　ポインタと配列

ポインタと配列には実は密接な関連があります。下記のプログラムを実行してみてください。

```
#include<stdio.h>
int main( void){
    int a[5] = { 5, 4, 3, 2, 1};
    int i;
    for( i = 0; i < 5; i++){
        printf( "a[%d] の値は %d 、アドレスは %d です¥n", i, a[i], &a[i])
;
    }
    return 0;
}
```

実行してくれればわかると思うんですけど、配列のアドレスが連番[ii]で振られていることがわかります。連番だとどうなるかというと次のような書き換えが可能となります。

```
#include<stdio.h>
int main( void){
    int a[5] = { 5, 4, 3, 2, 1};
    int i;
    for( i = 0; i < 5; i++){
        printf( "a[%d] の値は %d 、アドレスは %d です¥n", i, *(&a[0] +
i), (&a[0] + i));
    }
```

---

i　この関数はさらさら swap する気はないみたいだ

ii　宝くじみたくバラという選択肢はない

```
    return 0;
}
```

ポインタ風に書き直してみました。普通はこんな書き方しません。

第三節　関数を跨ぐ配列の受け渡し

関数を跨ぐ場合、大域変数ではない限り、ポインタを使わざる得ませんでした。しかし例えば配列の中身をソート[i]する関数を作る場合、一つずつ変数を渡していたらきりがありませんし、ソートする量によって関数を作り替えなければなりません。そういったときに配列を一度[ii]に渡してしまう方法があります。

```
#include<stdio.h>
int max5( int *array){
    int i;
    int max = 0;
    for( i = 0; i < 5; i++){
        if( max < array[i]) max = array[i];
    }
    return max;
}
int main( void){
    int a[5];
    int i;
    for( i = 0; i < 5; i++) scanf( "%d", &a[i]);
    printf( "一番大きな数は %d です¥n", max5( a));
    return 0;
}
```

こんな感じでしょうか？　配列が連番であり、最初のアドレスと個数が分かっているのならば、その配列の要素には自由にアクセスできます。max5 関数はポインタを使い、先頭のアドレスを受け取ったら、そこから 5 つ分のところをアクセスするようにしています。また、main 関数の printf 内で max5 関数を呼び出していますが、ここで注目して欲しいのは実引数であるアドレスの書き方です。ここでは単に「a」としか書いていません。配列の先頭アドレスを示す場合は本来なら「&a[0]」と書かなければなりませんが、単に配列名だけを書いた場合も配列の先頭のアドレスを示すという意味となり、このような書き方ができます。もし疑うのなら

```
printf( "&a[0] = %d, a = %d", &a[0], a);
```

という文を実行してみてください。同じ値を示すはずです。

しかし、この max5 という関数は配列数が 5 つのときではないと使用できません。その辺の改良の余地はは章末問題のほうに回します。

第四節　構造体へのポインタ

---

i　並び替えのこと。数の大きい順とか五十音順とか

ii　渡すはアドレス 1 つだけなんけどね

構造体へのポインタの話の前に構造体のことについて補足します。構造体も配列と同じようにまとめて変数が作れるという便利な物[i]ですが、配列と違い、同じ型の構造体ならば一括で全ての変数を代入できるという機能を有しています。

```
struct point{
    int x;
    int y;
}
int main( void){
    struct point pa, pb;
    pa.x = 3; pa.y = 4;
    pa = pb;
}
```

適当なサンプルすぎますが、main 関数の最後のところで構造体の代入が行われています。構造体の代入を使った代入を利用すると一度にまとめて多くの変数を代入することができます。これは関数間の引数においても同じことが利用できます。

ちょっと特殊な書き方なる構造体のポインタの説明をします。

```
#include<stdio.h>
struct point{
    int a;
    int b;
};
int swap( struct point *date){
    int buf;
    buf = (*date).a;
    (*date).a = (*date).b;
    (*date).b = buf;
}
int main( void){
    struct point pl;
    scanf( "%d, %d", &pl.a, &pl.b);
    swap( &pl);
    printf( "a = %d, b = %d¥n", pl.a, pl.b);
    return 0;
}
```

あえて分かりづらく書くとこんな感じです。アスタリスク、括弧、ドット演算子と３つも演算子を使っています。括弧が必要な理由は間接演算子よりドット演算子が優先順位が高いという決まりになっているからです。なので括弧を使って無理矢理間接演算子の優先順位を上げる必要があります。「*data.a」と書いた場合は data.a というポインタという意味になってしまい、私たち

---

i　この場合はまさに"オブジェクト"と言うべきか

が望む data ポインタで指した先のメンバ a という意味とは異なってしまいます。

しかし、きっと誰かが面倒だと考えたのでしょう、アロー演算子という便利なものが存在します。上記の swap 関数は次のように書き換えられます。

```
int swap( struct point *date){
    int buf;
    buf = date->a;
    date->a = date->b;
    date->b = buf;
}
```

このハイフンと大なり記号で表されたもの「->」[i]がアロー演算子と呼ばれるもので、構造体へのポインタという意味になります。通常はこちらのアロー演算子を使います。このアロー演算子は C++のクラス[ii]で嫌になるほど出てきます。

第五節　章末問題

1.三つの変数を引数に取り、小さい順に並び替える関数を作れ

2.渡された配列の平均値を戻り値とする関数を作れ

3.二つの配列とその要素数を引数として、片方の配列の内容を、もう片方にコピーする関数を作れ

4.構造体を使い、複素数の和、差、関、商の計算する関数を作れ(もちろん 4 つの関数を作ることになる)。但し、その関数内では計算のみを行い、計算した結果は構造体のポインタに出力するか、戻り値が構造体の形になるようにせよ

---

i　関係ないが、一般的な IME だったら"->"で変換すると矢印記号(→)になり便利

ii　構造体ともいう。C++においては構造体もクラスもほぼ変わらない

第三章　文字列

この章ではようやく数式から離れることができます。ゼミ A では華麗にスルーした文字の扱いに関してですが、何故スルーしたかというと難しいからです。文字列を扱う上で配列とポインタの知識をフルに使います。"hello　wolrd"と表示するだけでも裏ではポインタが隠されていたりと結構厄介なものです。

第一節　文字列と文字

まずは用語を説明します。C 言語では「文字列」というものと「文字」という二種類の分け方があります。文字というのは 1 つの文字だけを指し、シングルクォーテーション「'」で括られます。。文字列というのは文字の集まりで終端にヌル文字[i]が入ったものを指し、ダブルクォーテーション「"」で括られます。

```
printf( "character : %c, string : %s", 'a', "b");
```

上記の printf[ii]構文での「a」は文字、「b」は文字列です。どちらも見た目上、1 文字ですが、b のほうは終端にヌル文字と呼ばれるものが付いています。

```
sizeof( 'a');
sizeof( "b");
```

この二つの値を見れば、「a」の方は 1byte、「b」の方は 2byte と表示されるはずです。

ここで気づく人もいるかと思いますが、通常 printf で文字を表示するとき、第一引数もダブルクォーテーションで括りますが、これも文字列です。また DX ライブラリで画像をロードするときもアドレス[iii]をダブルクォーテーションで括りましたが、あれも文字列になります。

第二節　char 型配列

意識しないうちに文字列を扱っていることがわかりましたが、実際にどういう仕組みになっているか説明します。

```
#include<stdio.h>
int main( void){
    char str[256];
    str[0] = 'w';
    str[1] = 'i';
    str[2] = 's';
    str[3] = 'h';
    str[4] = '¥0';
    printf( "%s¥n", str);
    return 0;
}
```

---

i　null character。ナル文字という言う呼ぶこともある。

ii　printf の変換修飾子で文字は%c、文字列は%s を使う

iii　ここでの意味はファイルの場所のこと

まず char 型の配列を用意します。C 言語では一つの変数に一つの文字しか入れられません[i]。なので配列を使うことによって複数の変数を用いることにより文字列を実現します。配列の大きさは大きければ沢山の文字を入れることができますが、あまりに多いとメモリを無駄[ii]に使ってしまうので、ここでは 256 個の配列としました。そして七面倒ですが、C 言語では代入は一つずつしかできないのでサンプルのように一つずつ代入することになります。そして最後の文字のあと、必ずヌル文字を示す「¥0」を入れなければなりません。ヌル文字は文字列の終端を表す記号として使われている為、文字列ならば最後に必ずヌルを入れるという決まり[iii]になっています。

第三節　文字列に関する関数

文字列の代入が面倒ということはわかりましたので関数を使いましょう。文字列に関する関数は string.h というヘッダファイルの中に納められています。まずは文字列をコピーする関数から紹介しましょう。

```
#include<stdio.h>
#include<string.h>
int main( void){
    char str[256];
    strcpy( str, "wish you happiness");
    printf( "%s¥n", str);
    return 0;
}
```

ね、簡単でしょ？　先ほどのプログラムを書き換えてみました。strcpy 関数は第二引数の文字列を第一引数のアドレスに代入してくれます。また下記のように使用することもできます。

```
#include<stdio.h>
#include<string.h>
int main( void){
    char str1[256];
    char str2[256] = "wish you happiness";
    strcpy( str1, str2);
    printf( "str1:%s, str2:%s¥n", str1, str2);
    return 0;
}
```

char 型変数から char 型変数へのコピーです。strcpy 関数はコピーをする関数なので実行後でも元に入っていた変数に影響はありません。

さらりとこのプログラムで str2[256] = "wish..."と書いていますが、この書き方をできるのは変数の宣言時のみです。それ以外は strcpy 関数など[iv]を使わないと不可能です。

---

i　面倒だと思ったそこの貴方！！　Perl を勉強しよう！！

ii　さっき言ってたことと矛盾としているような気がするのは気にしない

iii　逆に、Pascal という言語は配列の先頭に文字列の長さを格納しているとか

iv　sprintf とか

次に二つの文字列が同値かどうか判断する strcmp 関数です。

```
#include<stdio.h>
#include<string.h>
int main( void){
    char str[256];
    scanf( "%s", str);
    if( strcmp( str, "sweets") == 0) printf( "(w)");
    else printf( str);
    return 0;
}
```

scanf で文字列を読み込み、その文字列が「sweets」だったら「(w)」と、それ以外なら入力された文字列を表示するプログラムです。上記の場合は、変数と定数の比較ですが、もちろん変数と変数の比較もできます。strcmp 関数は二つの文字列が同値のときに 0 を返します。

```
if( !strcmp( str1, str2)) { ...}          //同値の時の処理
```

このように論理否定演算子である「!」を付けるとタイプする文字が少なくなります。

stdio.h にも文字列に関する関数が用意されています。sprintf 関数と sscanf 関数です。この両者は名前の通り printf と scanf に非常によく似ています。通常の printf、scanf が画面に出力、キーボードから入力するのに対し、sprintf は文字列への書き出し、sscanf は文字列からの入力となります。

```
#include<stdio.h>
int main( void){
    char str[256];
    int i;
    for( i = 0; i < 10; i++){
        sprintf( str, "今は %d ループ目です", i);
        puts( str);          //printf( "%s¥n", str)と同じ意味
    }
}
```

int 型の数字を文字列に変換できるという機能を持っています。この機能を使って、連番で振られているファイルを読み出すときなど、連番の文字列が必要なときに使えます。

第四節　char 型ポインタ

配列を毎回宣言するのは面倒ですよね。こんな時はポインタを使いましょう。

```
#include<stdio.h>
int main( void){
    char *str;
    str = "wish you happiness";
    printf( "%s¥n", str);
    return 0;
}
```

ポインタを使って文字列を扱います。char 型ポインタ str は"wish..."のアドレスを代入する、と記述されていますが、いったい何処のアドレスなのでしょうか？　まぁどっかです。イメージとしてはその辺のメモ帳に適当に書いたといった感じです。一時的に文字列を扱いたいときに使います。ここで先ほどの配列を利用したときと違うのは直接文字列をポインタに代入できる点です。下記のプログラムは５行目でエラーがでます。

```
int main( void){
    char *str_p;
    char str_v[256];
    str_p = "wish you happiness";    //祈りはコンピュータに届いた
    str_v = "wish you happiness";    //祈りはコンピュータに届かなかった
    return 0;
}
```

この char 型ポインタは関数の引数としてよく使われます。printf 関数や strcpy 関数がその例です。printf 関数の内部では私たちアドレスとして渡す引数を全てポインタで扱っています。普通の変数として扱ってしまうと扱える文字列の長さが配列によって制限されてしまうからです。ポインタとして扱えば、外部に文字列本体を置くことができ、文字列の長さ関係なく文字列を扱う事ができます。このような芸当ができるのは終端にヌル文字が付いているからこそです。ヌル文字がなかったらどこまでが文字列なのかがポインタにはわかりません。ということはヌル文字が付いていなかった場合は、わかりますよね？

第五節　関数の引数としての文字列

関数は一定の決まった処理をまとめるという効果があります。例えば、ある文字列の中から指定したワードを検索する、といった関数はなかなか使える関数ではないのでしょうか？

文字列を検索する関数は少し難しいのでまずは文字列の長さを得る関数を作りたいと思います。文字列の長さというのは散々説明した通り、ポインタの始めの位置からヌル文字まで[i]の長さです。

```
#include<stdio.h>
int str_length( char *str){
    int len;
    for( len = 0; str[len] != '¥0'; len++);
    return len;
}
int main( void){
    char str[256];
    gets( str);
    printf( "%s の文字列の長さは %d です¥n", str_length( str));
    return 0;
}
```

i　ヌル文字は含めない

gets という見慣れない関数がありますが、scanf("%s", str)とほぼ同じもの[i]と考えてください。

上記のプログラムですが、str_length 関数は char 型のポインタを引数にとっています。そして for 文を使い○番目の配列、一番目の配列……と順々に配列の中身を調べていき、ヌル文字がないか調べています。調べた配列がヌル文字ではない場合は len をインクリメント[ii]し、あった場合はブレイクします。また for 文の使い方も普段は中括弧を使っていくつか文を書くことが多いと思いますが、for 文だけで事が足りるので、最後にセミコロンを付けて文(ステートメント)として完結させます。

あと配列を宣言していないのに配列を使っているのは奇妙だと思う場合は

```
int str_length( char str[]) { ...}
```

このような書き方もありです。主に気分によって使い分けてください。

第六節　章末問題

1.strcpy 関数を作れ

2.strcmp 関数を作れ

3.strcat 関数[iii]を作れ

4.キーボードから文字列を読み込み、その文字順とは逆に画面に出力するプログラムを作れ (ex: abcdef → fedcba)

5.二つの文字列を引数に取り、片方の文字列の中に、もう片方の文字列が存在する場合は 1 を返す関数を作れ

6.下記の Caesar 暗号[iv]を解読するプログラムを作れ。ちなみに sugar という文字列が入っているらしいです。

"nygeyagckx,g,yg?ovvg?aqk!gck,o!gpy!g,rog!o?,gypgeya!gvspojgy!gnygeyagckx,g,ygmrkxqog,rogcy!vni"

使用される文字の列"abcdefghijklmnopqrstuvwxyz !?,"(z と!の間にスペースが入ります)

---

i　scanf を使った場合はスペースやタブは区切り文字(デリミタ)として扱われるため、"wish you happiness"という文字列は三つの文字列として扱われる

ii　i++のこと。i--はデクリメントという

iii　二つの文字列を引数とし、片方の文字列の末尾にもう片方の文字列をつなげる関数

iv　例えば hello という文字列を暗号化するとき、アルファベット順に一文字ずつずらして、igmmp という文字列にする具合。この問題の場合、何文字ずらすかは示されていないので、sugar という文字列を頼りにするしかない