関数と、ファイルの保存・読出し、構造体

前々回の RPG（ドラゴンクエストもどき）は、結構みんな頑張っていたようでしたので、今回はその発展形を題材にすることにしました。（と昨年書きました。）
皆さん、ずいぶん逞しく成長して来ていると思います。

エラー対応は、「慣れ」が大切ですが、面白さを覚えて「慣れる」までに疲れて面倒くさくなってしまうのが通例なので、「エラー」に悩まされている時は、遠慮しないで手を挙げてください。
頑張った内容は、「スキル」として自分の能力アップにつながっています。自信を持ってください。多くの人がかなり実力をつけています。
さて、今日の教材は、ほとんど「初心者向け」の配慮をせずに「使える機能は普通に使う」ようにして、書きました。複数の人間が「分割して作成する」という処理はしていませんので、全部が一つのプログラムファイルに収まっていますが、それ以外は実践レベルの内容です。(※変数のスコープとか、組み込んでいない要素も多数ありますが・・・)

ということは、「完璧に理解して、修正を行う」というよりも、まず読んで、目を慣らして、雰囲気をつかむのが、主眼です。「長文読解」の教材ですが、雰囲気がわかれば OK です。

ある程度雰囲気がつかめたら、そこから、「課題」の修正に挑戦してみてください。
最初は、関数について学習しますが、ファイルの保存と構造体については読み方、書き方のコツをつかんで下さい。
どちらも、 .（ピリオド）や *(ポインタ演算子)など、記号文字の使い方、書き方などが重要です。

関数

実は、関数は初登場ではありません。これまで、どんな関数が出て来たかと言うと、

printf()　scanf()　rand()　srand()　_getch()

などがあります。

printf()は、関数ですが、””（二重引用符号）で囲まれた範囲を基本として表示しながら、%d, %s, %c などを設定すると、その「書式」に従って、数字や文字などを表示できました。

その「書き方の決まり」や「約束事」などは、関数ごとに異なっています。
標準関数ではなく、自分で定義した「ユーザ関数」も、既に何度も使ってしまっています。教科書を使って、おさらいします。

それが、前々回の以下のプログラムです。　（438 行目から 461 行目付近）

```
/*
 * 攻撃力を計算する
 */
int calc_attack( int attack_power )
{
	int p = rand()%31 + 70; // 70% ～ 100%
	int attack = attack_power * p / 100;

	switch( your_weapon ){
	case 0:
	default:
		return	attack;
	case 1:
		return	attack * 3 / 2;		// 1.5 倍：ナイフ
	case 2:
		return	attack * 2;		// 2 倍：竹やり
	case 3:
		return	attack * 3;		// 3 倍：胴の剣
	case 4:
		return	attack * 5;		// 5 倍：勇者の剣
	}
}
// ============================================================
```

関数の「名前」は、 calc_attack です。
「引数」は、一つだけ持っています。メインのプログラムから、「 attack_power」という変数の「値」を渡されて、それに対して攻撃力を計算しています。
この関数の目的は、「パワー」から「攻撃力」を計算することで、２段階で計算しています。
第１段階では、乱数で「attack_power」の 70～100% の値を計算します。
第２段階では「武器」（ your_weapon ）の値に応じて、「攻撃力」を倍加しています。
そうして計算された値は、以前のバージョンでは your_attack という変数に代入しました。今回は、「普通の関数」の書き方で、return 文で戻しています。

```
int calc_attack( int attack_power )
```

と書いてあります。この関数は int 型なので、int で結果を戻して、本体側で受け取っています。前回のプログラムでは、この関数内で your_attack に代入していました。

なぜ、関数を使うのでしょうか？

関数を使うことで「本体部分」がスッキリとして、メインのプログラムの流れを理解しやすくなる、ということがあります。

今回のプログラムは、以前のプログラムに対して若干「パワーアップ」しています。

※　ステージを５段階に分けた。
※　ステージごとに、登場するモンスターを分けた。
※　武器や防具の種類を増やした。
※　ゲームを保存したり、保存したゲームを読み出せるようにした。

などです。約 800 行のプログラムになりました。

プログラムの全体構造

198 行目から 408 行目が、メインの、最も大きなループです。
ここでは「竜の王冠」を手に入れるまで、反復処理を行います。

ループの入り口でメニューを表示させて、メニューによって処理が分かれます。
その switch / case が、208 行目から始まり、メインのループ全体にかぶさ

っています。
ここで、処理番号１～５は、それぞれが「関数」として実行されるように構成されています。
10 行目から 19 行目は、「プロトタイプ」と呼ばれ、その関数の「型」や引数の種類を指定します。以下の関数について簡単に説明しましょう。

```
extern   int        menu( void );
extern   int        set_monster( int );
extern   void       save_game( void );
extern   void       go_shop( int );
```

複数の引数を渡すことができますが、ここでは上記のような４種類の関数を使いました。
関数「menu」では、引数は渡さず（ void は、型がないことを意味します）選択されたメニューの番号をreturn しています。521 行目から536 行目までで、メニューの表示と、数値の処理を行って返しています。

関数 set_monster では、登場させるモンスターを乱数で決めています。
ステージごとに、登場するモンスターに違いがあります。
この、どのステージに、どのモンスターを登場させるかを、stage_monster という二次元配列で設定してあります。使っているのは、593 行目ですが、定義は 52 行目にあります。ここでは、引数として int　でステージ番号を手渡し、そのステージで登場できるモンスター番号のリストを作成し、次に、その作成されたリストの中から乱数で登場させるモンスターを選んでいます。
stage_monster[モンスター番号][ステージ番号]が true なら、そのモンスターがそのステージに登場することを意味し、false なら登場しないことを意味します。

関数「save_game」では、引数も戻り値もなく、ゲーム全体を制御するパラメータを保存しています。
通常は、こうした関数では「保存成功」なら true を、「保存失敗」なら false を返すようにプログラムしますが、今回は失敗してもそのまま続行しています。

関数「go_shop」では、ステージごとに購入可能な品物の種類を変えています。
126 ～ 137 行目で定義している stage_item という二次元配列で、この組み合わせを制御しています。但し、何かを購入した処理はすぐにその場で処理

されているため、return で値を戻すということはしていません。void 型です。
こうした関数で、メニュー分岐の個別の処理を行い、全体構造を見やすくしています。
ステージ選択では、「持っているアイテム」に応じて選択できるステージを分けています。
258 行目から、404 行目までは、それぞれの「ステージ」ごとのループで、「鍵」を手に入れるまでは抜けることができません。

このゲームプログラムでは、ほとんどの変数を「グローバル変数」として定義しています。実は、グローバル変数はあまり多用するべきではなく、本格的にプログラミングを学ぶならば「変数のスコープ」を意識しなければならず、関数の引数はもっと増える場合が多いのです。但し、皆さんは臨床工学なので、スコープについては学びません。
授業では扱わないけれど、ステップアップすると少し書き方が変わると理解して下さい。

【報告課題（１）】

それぞれのステージで、「これ以上続けられない」と思った時に、そのステージから「脱出」するためには、プログラムのどの行をどのように書き換えたらいいでしょうか？

【ヒント】ステージごとのループ（258 行目～405 行目）では、stage_play（最もレベルの高いステージに進むのではなく、以前クリアして「鍵」を持っているステージを選んでプレイしているモード）の場合と、「鍵を持っていない間」は、反復するようにしています。
ここで、stage_play の時だけ投げている質問を「いつでも」確認するように書き直せば、修正完了です。
278 行目で、逃げるかどうか確認していますから、ここで「脱出」を決めてもいいでしょう。体力が落ちている場合は 287 行目でドリンクを飲めるようにしました。この場合は戦闘メニューの表示を変えて、入力を処理するという修正になるでしょう。

何行目をどのように修正したら、ステージから脱出出来るようになるか、
考え方を説明し、修正前と修正後のプログラムを示し、確認表示が出て、選択した結果でループを抜けた実行画面を添付して下さい。（配点：３点）

戦闘は、290 行目から 391 行目のループです。
相手の攻撃は、294 行目から 315 行目（倒された場合も含む）
あなたの攻撃は 317 行目から 324 行目、
相手を倒した場合の処理が、326～390 行目です。

ステージごとに、倒したモンスターによって、手に入れられるものが変化しています。その点については、プログラムを読んでください。

構造体

今回初めて使われたのは、構造体です。モンスターの定義に構造体を使いました。22 行目から 39 行目です。

モンスターの名前、プログラム上の識別番号、攻撃力、防御力、倒した時に得るゴールドの値、倒した時に得る経験値、などがモンスターごとに定義されています。

複数のデータを一度に組み合わせるため、「構造体」は使われます。
配列をたくさん用意する、という方法もありますが、その場合だと、追加や削除があった時に、修正漏れがおきて、順番がずれるなどのバグの原因になりやすいので、構造体を用います。

```
struct 構造体名 {
        変数の型　変数名;
        変数の型　変数名;
                ;
                :
}
```

という書き方が基本形です。発展形として、bit ごとの定義がありますが、組込み系プログラムや、機器制御など以外にはあまり使いません。

この例では、構造体の定義と、データの配列の設定を一度に行いました。

```
// モンスター定義構造体
struct Monster {
        int     id;             // 識別番号
        char    *name;          // 名前
        int     min_attack;     // 攻撃力初期値・最小値
        int     max_attack;     // 攻撃力初期値・最大値
        int     defense;        // 防御力初期値
        int     gold;           // ゴールド所持初期値
        int     experience;     // 取得可能な経験値
}
```

この構造体は、上から順に id, name, min_attack, max_attack, defense, gold, experience となっています。
41 行目から 48 行目まで、８種類のモンスターが定義されています。
先頭のスライムは、0, ″スライム″, 10, 15, 15, 3, 2 となていますが、これは、プログラム中での ID が 0, 名称がスライム、このモンスターの戦闘能力は 10 から 15 の間で、このモンスターを倒すと３ゴールドが手に入り、このモンスターを倒した時の経験値は２増加する、ということを意味しています。
今回の例では、int と char *型だけでしたが、float や double などが使われることもあります。
これが使われている部分の例は、267～269 行目にあります。

```
monster_power = rand() % (monster[monster_id].max_attack-
                monster[monster_id].min_attack+1)  +
                 monster[monster_id].min_attack;
```

このモンスターのパワーは、乱数で最小値から最大値の間になるように計算しています。（前回のバージョンでは、プログラム中に数字で書いていた部分を、構造体定義の一覧表の部分に移動した形になっています。）

なんだか、難しく感じるかも知れませんが、難しく考えずに、「セットメニュー」で、「今日のおかず、今日のみそ汁、今日のごはん/パン、今日の漬け物」などが４つ１セットとして定義され、それが毎日入れ替わるけれども、その入れ物は変わらない、それが構造体だと思ってもらえればいいでしょう。

ファイルへの保存と読み出し

今回は、「ゲーム仕立て」です。ですので、ファイルへの保存は、ゲームの中間経過を保存する、という作業をどうプログラムするかで教材としました。
同様にして、ファイルからの読み出しも、「ゲームを途中から再開する」という形になっています。

ゲームの保存は、以下のようになります。

```
/*
 *  ゲームの保存
 */
void    save_game(void)
{
          int     i;
          FILE    *fp;
          int           errno;

          errno = fopen_s( &fp, "gemesave.gsv", "w");
          if ( errno!=NOERROR ) return;
          fprintf(fp, "%d,%d,%d,%d¥n", your_hitpoint, your_power,
                    your_max_hitpoint, your_max_power);
          for (i = 0; i<NUM_STAGES; i++){
                    fprintf(fp, "%d", (have_pass[i] ? 1 : 0));
                    if (i<NUM_STAGES - 1)     fprintf(fp, ",");
                    else                      fprintf(fp, "¥n");
          }
          fclose(fp);

          printf("ゲームを保存しました。¥n");
          Sleep(1000);
}
 // ================================================================
```

最初にファイルを開きます。ここで、「書き込みモード」(“w”)か、「読み出しモード」(“r”)かを指定しますが、save は保存なので「書込み」です。
「書き込みモード」でファイルが存在しない場合には、新規作成されます。
548 行目の fopen_s　と、557 行目の fclose までが、書き込みのためのプログラムです。
fopen_s の第１パラメータは、ファイルポインタで、「アドレス渡し」指定をしています。そのため、&がついていますが、ここはこういう書き方だと思っていて下さい。第２パラメータはファイル名です。プログラムでファイルを指定する時は、拡張子まで省略せずに記載しなければなりません。第３パラメータの”w”は「書き込み」を意味します。

書き込みでは、fprintf が使われますが、どこかで見たことがありませんか？

画面に文字列を表示する printf に、f がついた関数です。関数名に f がついていて、第一パラメータが fp（file pointer）である以外は、printf と同じ使い方ができます。

保存を行っている最初の文は、

```
fprintf( fp, "%d,%d,%d,%d¥n", your_hitpoint, your_power,
            your_max_hitpoint, your_max_power );
```

です。%d が４回（10 進整数が４つ）カンマで区切られながら並んでいます。

your_hitpoint, your_power, your_max_hitpoint, your_max_power

の値が、それぞれ何を意味しているかわかりますか？

問題は、この次からです。

```
for( i=0; i<NUM_STAGES; i++ ){
    fprintf( fp, "%d", (have_pass[i] ? 1: 0) );
    if( i<NUM_STAGES-1 )      fprintf( fp, "," );
    else                      fprintf( fp, "¥n" );
}
```

この２行目、 `(have_pass[i] ? 1 : 0)` という記述があります。この行は、?と：がワンセットになって、一つの演算子になっています。

条件式　？　（条件が真の場合の値）　：　（条件が偽の場合の値）

ですから、have_pass[i]が、もし true なら１が、false ならば０が書き込まれます。１や０は integer なので、%d　（十進整数）としてファイルに書き込んでいます。

次の if 文では、i が「最後」のステージでない場合には ,（カンマ）を、最後のステージの場合には ¥n（改行）を書き込んで、１行になるようにしています。

こうして保存された内容は、例えば以下のようになります。

```
1 106,106,106,106↓
2 1,0,0,0,0↓
3 [EOF]
```

ファイルの読み出しも、保存の場合とよく似ています。

```
/*
 *  ゲームの読出
 */
void    load_game(void)
{
        FILE    *fp;
        int     a, i;
        int     errno;

        errno = fopen_s(&fp,"gemesave.gsv", "r");
        fscanf_s(fp, "%d,%d,%d,%d¥n", &your_hitpoint, &your_power,
                &your_max_hitpoint, &your_max_power);
        for (i = 0; i<NUM_STAGES; i++){
                fscanf_s(fp, "%d", &a);
                if (i<NUM_STAGES - 1)    fscanf_s(fp, ",");
                else                     fscanf_s(fp, "¥n");
                if (a == 1)    have_pass[i] = true;
                else            have_pass[i] = false;
        }

        fclose(fp);

        printf("ゲームを読み出しました。¥n");
        Sleep(1000);
}
 // ============================================================
```

まず、最初の fopen のファイル名の次のパラメータが、”r”になっています。これは、read(読み込み)でファイルを開くことを意味します。

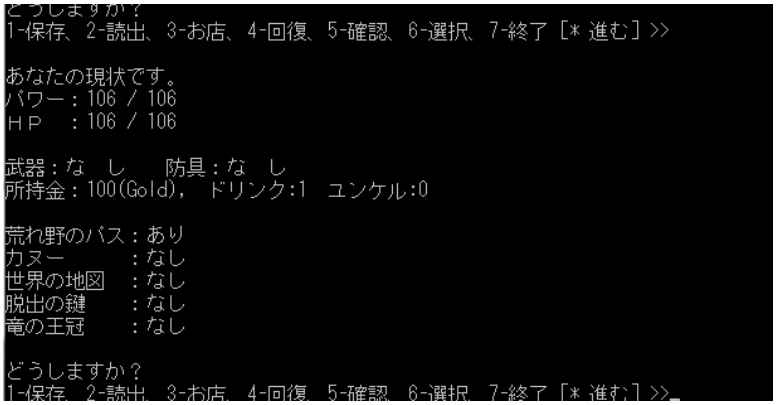
書き出し（保存）は fprintf でしたが、読み込みは fscanf_s で実行できます。使い方は、scanf と同じで、書式指定を行ってデータを読み込みます。
読み込んでいる変数名には&をつけてアドレス指定をしています。ここを間違える人が多いので、特に気をつけて下さい。

そして、読み込む順番は、書き出した順番と全く同じにします。順番が狂うと、数字の意味が変わってしまいますので気をつけて下さい。
最後の have_pass は、読み込んだ値が 1 ならば true、 0 ならば false となるように読んでいます。

Visual Studio の「警告」で、fscanf は危険だから fscanf_s を使え、というメッセージがありました。警告がうっとうしいので、素直にその指示に従っています。教科書的には、scanf_s は scanf で、fscanf_s は fscanf です。

本当に、きちんとファイルに保存し、きちんとファイルから読み出せているのでしょうか？それを確認するために、menu の５番目に、「確認」というコマンドを入れました。
RPG では、今現在の HP（ヒットポイント）がどの程度か、把握していないと簡単に倒されてしまいます。ですので、どんな装備で、ドリンクはあといくつ残っていて、などという項目は、時々確認したくなります。そのためのコマンドです。

このコマンドを実行すると、画面に次のようなメッセージが表示されます。



【報告課題（２）】

保存していないパラメータもあります。「武器」や「防具」を買っても、その情報は保存されないので、ゲームを終了して読み込むと、また何もない状態から始まります。また、ゴールドや、ユンケルも、保存されないために、初期値にリセットされます。（ゴールドは 100、ドリンクは 1 になります。）

この情報を保存するためには、プログラムをどう修正したらいいでしょうか？プログラムが書かれていて、動作の検証方法の報告があって、プログラムの説明があって、配点５点とします。

文法についての説明は、以上で終わりです。

【報告課題（３）】

二次元配列に関する課題です。

こうしたゲームでは、シナリオによって、様々な「ステージ」や、「モンスター」、「アイテム」を登場させています。ステージを増やす際には、モンスターを増やす際には、また、アイテムを増やすためには、どんなプログラム上の注意が必要か、プログラム構造を読んで、注意点や方法をまとめ、可能ならば実際にステージや、モンスター、アイテムなどの追加を行って、動作検証まで行ってください。（モンスター、ステージ、アイテムそれぞれの追加ごとに、最大３点まで加点します。）

> NUM_MONSTERS など、配列のサイズを指定している部分に注目し、配列の個々のデータの設定を、「矛盾がないように設定する」ということが大切になります。
>
> 手順としては、「変数名」で検索し、使われている場所の行番号を控え、追加に「漏れ」がないように書いて下さい。漏れがあった場合には減点しますが、挑戦していたら１点は出します。

【発展課題】

プログラム構造の変化を伴う改造を行ってください。

（モンスターの名前を変えただけ、ヒットポイントや、経験値、ゴールドの値、アイテムの値段などの数字を変えただけ、というのは、ダメです。）

この部分をこう変えたら、「宿屋に宿泊」して、ヒットポイントが全回復できるようになった、とか、全面的にこういう改造をしたら「魔法攻撃」ができるようになった、とか、ある局面では一定の確率で「武器を紛失」することになった、とか、プログラムの構造（論理）の改変を伴う改造を行ってください。

※　こうしたいと思った。

※　プログラムをこう変更したら、こうなった。

※　それは、こうやって検証した。

この三つを明らかにして、報告してください。難易度に応じて加点を行います。

前回に RPG のゲームを素材とした時と同様に、今回も多くの「発展形」が考えられます。

長文読解がベースですが、このプログラムを普通に「読める」ようになったら、あとは文法書で確認しながら、どんなプログラムでもどんどんと読んでいけるでしょう。
そうしたら、「こんなことをやってみたい」という自分で見つけたテーマをプログラムするのにいつでもできるようになります。

「入門編」の卒業試験だと思って取り組んでみてください。

頑張った人には、それだけの実力がついていますので、自信を持ってください。

来週から「アルゴリズム」を中心として、「プログラミングの考え方」の方向に流れを変えて行きます。

なお、次回からの教材では「プログラムのサンプル」を一切配布しません。また、配布教材にもヒントは書いてあってもプログラム自体はほとんど書きません。

この先は、ごく普通の「プログラミング演習」になります。内容的には逆に退屈になるかも知れませんが、基礎体力がついていますので、「丁寧に」行えば確実に課題報告ができると思います。

これまで、「どんな処理をする時に、どんなプログラムが書かれていたか？」を思い出してみてください。