

2009 年度 プログラミング演習 3 配布資料

担当：北野 (D1)、谷口 (D2)

◎ スケジュール、課題

《第 1 部》

- 第 1 回 乱数を使ったプログラム：random()、srandom()
- 第 2 回 サイコロゲームの作成：ソースプログラムの文書化
- 第 3 回 トランプカードの表現：ヘッダファイル、ライブラリ
- 第 4 回 トランプカードの操作：データ構造
- 第 5、6 回 トランプゲームの作成：分割コンパイル、make コマンド

《第 2 部》

- 第 7～11 回 グループワークによるデータベースプログラムの作成

《第 3 部》

- 第 12～15 回 自由課題

◎ 成績評価

- ・ 課題への取り組み姿勢 (10%)
- ・ 第 1 部演習課題 (30%)
- ・ 第 2 部グループ課題レポート (30%)
- ・ 第 3 部自由課題レポート (30%)

但し、10 回以上の出席と指示された全ての提出物の提出を必須条件とする。

第1回 乱数を使ったプログラム

コンピュータは通常、指示された実行命令の通りに処理を行うが、乱数を利用することで不確定性をもつ処理を行わせることもできる。コンピュータで生成される乱数を疑似乱数（真の意味の乱数ではない）といい、関数 `random()` を呼び出すことで、疑似乱数を利用することができる。

関数 `random()` は 0 から $2^{31}-1$ までの整数を“ランダムに”返す関数であり、その呼び出し例は次の通りになる。

《プログラム 1-1》

```
#include <stdio.h>
#include <stdlib.h>    //random()を呼び出す時に記述するヘッダファイル

int main()
{
    long randNum;

    randNum = random();    //乱数を1つ生成し、変数nに代入
    printf("random number = %ld\n", randNum);

    return 0;
}
```

プログラム 1-1 を実行すると、毎回同じ値が出力されることがわかる。これは、`random()` が真にランダムな値を生成しているのではなく、ある法則に従って生成した値を返しているからである。実行の度に異なる値を生成させるためには、乱数の種 `seed`（乱数の初期化に用いられる、ただし初期値そのものではない）を与えることで実現できる。`seed` の設定は関数 `srandom()` を用い、次のように呼び出す。

《プログラム 1-2》

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long randNum;
```

```

unsigned seed;

printf("乱数の種 seed を入力してください：");
scanf("%u", &seed);
srandom(seed);    //乱数の種を設定
randNum = random();
printf("random number = %ld¥n", randNum);

return 0;
}

```

プログラム 1-2 実行時に異なる `seed` を入力すると、異なる乱数値が表示されることがわかり、また、同じ `seed` を入力すると、必ず同じ乱数値が表示されることがわかる。実行の度に、異なる乱数を生成させたい場合、システムのもつ時刻を取得する関数 `time()` を用いることができる。

《プログラム 1-3》

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>    //time()を呼び出す時に記述するヘッダファイル

int main()
{
    long randNum;

    srandom(time(NULL));    //time()で取得した時刻を乱数の種として設定
    randNum = random();
    printf("random number = %ld¥n", randNum);

    return 0;
}

```

このプログラムでは、実行の度に標準入力から `seed` を入力することなく、毎回異なる乱数値が出力されることがわかる。

【課題 1-1】 プログラム 1-3 を実行して動作を確かめよ。

プログラム 1-3 では、`random()` の返す値 ($0 \sim 2^{31}-1$) をそのまま表示しただけであるが、その値に演算を施すことでサイコロふりを表現することができる。いわゆる六面体のサイコロを振った場合、その目は 1、2、3、4、5、6 のどれかがランダムに出る。

【課題 1-2】 変数 `n` にサイコロの目の値 (1~6) が代入されるようにプログラム 1-3 を修正し、サイコロふりをシミュレートせよ。これは、

```
randNum = □+random()%□;
```

とし、□に適当な値を入れることで実現できる。

【課題 1-3】 サイコロの目を返す処理を関数 `rollDice()` として括りだせ。`main()` 内では、

```
randNum = rollDice();
```

として呼び出したとき、`n` にサイコロの目が代入されるようにせよ。

[補足] 疑似乱数の生成には、0 から `RAND_MAX` (`stdlib.h` に定義されている記号定数) の間の整数を返す関数 `rand()` が従来用いられてきたが、アルゴリズムの性質により何度も用いると周期的な数の列を生成するため、乱数列としてはあまり好ましくない。上記の `random()` は `rand()` を改善したものであり、よりランダムな数の列を生成することができるため、最近ではこちらがよく用いられる。

第2回 サイコロゲームの作成

前回の課題で作成したサイコロの目を返す関数 `rollDice()` を使い、カジノで有名なブラックジャックをサイコロで行うゲームをプログラムする。このゲームは、ディーラー（親）とプレイヤー（子）が対戦し、振ったサイコロの目の合計（ポイント）が 21 に近い方が勝ちとなる。

《手順》

1. サイコロを振る

プレイヤー、ディーラーとも 2 回ずつサイコロを振る。プレイヤーのサイコロの目は 2 回ともオープンにし、ディーラーの 1 回分は見えないようにする。

2. プレーヤーの番

プレイヤーはヒット（もう 1 回サイコロを振る）かスタンド（その時点のポイントで勝負）を選択する。21 を越えない限り何回でもヒットすることができる。但し、21 を越えると直ちにプレイヤーの負けとなる。

3. ディーラーの番

プレイヤーがスタンドすれば、次にディーラーがサイコロを振る。ディーラーの場合、ディーラーの意思でサイコロを振るかどうかを決めるのではなく、ポイントが 17 以上になるまでサイコロを振らなければならない。また、17 以上になったら、その後はサイコロを振ることはできない。

4. 勝敗の決定

- ・ ディーラーが 21 を越えた場合、プレイヤーが 21 を越えていなければプレイヤーの勝ち
- ・ ディーラーが 21 を越えていなければ、21 に近い方が勝ち。同じポイントであれば引き分け。

【課題 2】 以上の手順を 1 ゲームとし、コンピュータがディーラー、ユーザがプレイヤーとして実行できるようにプログラムせよ。但し、ここでは、1~10 の目をもつ 10 面体のサイコロであるとする。

《実行例》

```
% ./blackjack
```

ゲームを開始します。

サイコロを振ります。

ディーラー>目は1と??

プレイヤー>目は3と8。ポイントは11。

プレイヤーの番です。

ヒット or スタンド?[ヒット=1 スタンド=0]: 1

プレイヤー>目は5。ポイントは16。

ヒット or スタンド?[ヒット=1 スタンド=0]: 0

ディーラーの番です。

ディーラー>目は1と9。ポイントは10。

ディーラー>目は5。ポイントは15。

ディーラー>目は4。ポイントは19。

プレイヤーのポイントは16、ディーラーのポイントは19。

ディーラーの勝ちです。

%

【課題2': プログラム作成上の注意】

ソースプログラムを読みやすくする為に、コメント文を丁寧に記述することは重要ではあるが、それ以前にプログラムそのものを読みやすくしなければならない。その1つの方法として、処理の意味が一目見てわかるような変数名、関数名を用いることを心がけるべきである。

例えば、

```
n = func();
```

```
x += n;
```

とするより、

```
spotPlayer = rollDice();
```

```
pointPlayer += spotPlayer;
```

とすれば、コメント文を添えなくとも、変数名、関数名からその処理が理解できる。課題2のプログラムがこのようになっていない場合、変数名、関数名を適切なものに修正せよ。

第3回：トランプカードの表現

トランプの各カードはスーツ（スペード、ハート、クラブ、ダイヤ）とフェース（A、2、3、…、J、Q、K）の2つの属性の組み合わせで表される。複数の属性により1つのデータが表される場合、そのデータをC言語で扱うには構造体が適している。

【課題 3-1】 char *face と char *suit をメンバとする構造体 struct card 型を定義し、さらに typedef により、Card 型を定義し、次のプログラムを完成させよ。

《プログラム 3-1》

```
#include <stdio.h>

//ここに構造体定義

int main()
{
    Card oneCard;

    oneCard.face = "A";
    oneCard.suit = "ハート";
    printf("カード: %s の%s\n", oneCard.suit, oneCard.face);

    return 0;
}
```

プログラムを開発するにあたり、作成する関数を機能ごとに分類し、ソースファイルを分割してコーディングする方が効率よい場合がある。この時、分割された複数のソースファイルに共通して用いられる宣言や定義があるなら、各ソースファイルで逐一それらの宣言、定義を記述するのではなく、それらの宣言、定義を記述したヘッダファイルを作成して、各ソースファイルでインクルードするのが普通である。

ヘッダファイルには、記号定数の定義、構造体の定義、関数プロトタイプなどが記述されることが多い。標準ライブラリヘッダファイル（C言語開発環境で用意されている `stdio.h`、`stdlib.h` など）をインクルードする時には、

```
#include <stdio.h>
```

とするが、プログラマが定義したヘッダファイルの場合は、

```
#include "card.h"
```

と記述すればよい。

【課題 3-2】 card 構造体を定義したヘッダファイル card.h を作成し、次のプログラムを動作させよ。

《プログラム 3-2》

```
#include <stdio.h>
#include "card.h" //ユーザで作成したヘッダファイル

int main()
{
    Card oneCard;

    oneCard.face = "A";
    oneCard.suit = "ハート";
    printf("カード: %s の%s¥n", oneCard.suit, oneCard.face);

    return 0;
}
```

【課題 3-3】 52 枚全てのカードを Card 型構造体の配列 deck により表す。カードデッキ deck を初期化（配列の全ての要素の各メンバに対し、値を設定）する関数 initCardDeck() を作成し、次のプログラムを完成させよ。

《プログラム 3-2》

```
#include <stdio.h>
#include "card.h"

void initCardDeck(Card *deck)
{
    char *face[13]={"A","2","3","4","5","6","7","8",
                  "9","10","J","Q","K"};
```

```
char *suit[4]={"スペード","ハート","クラブ","ダイヤ"};
int i;

//配列 face と suit を利用し、配列 deck を初期化する。

}

int main()
{
    Card deck[52];

    initCardDeck(deck);    //カードデッキを初期化

    //初期化後の deck を全て表示する。

    return 0;
}
```

【課題 3-4】 関数 `initCardDeck()` を別のソースファイル `card.c` に記述し、プログラムを2つ、`main()` を含むプログラム全体の流れのみをコードする `main.c` と `Card` 構造体に関する処理をコードする `card.c`、に分割して作成せよ。

第4回 トランプカードの操作

カードゲームを行うには、カードデッキに対しシャッフル（切る）とディール（配る）などの操作が必要になる。

【課題 4-1】 カードデッキをシャッフルする（ランダムに並び替える）関数 `shuffleCardDeck()` を作成して `card.c` に加え、カードデッキを初期化後、シャッフルしてその結果を出力せよ。

《関数 `shuffleCardDeck()`》

ライブラリ	<code>card.h</code>
用法	<code>void shuffleCardDeck(Card *deck);</code>
処理内容	Card 型配列 <code>deck</code> の各要素をランダムに並び替える

【課題 4-2】 配られた 1 枚のカードを手札に加え、手札を更新する関数 `updateHand()` を作成して `card.c` に追加し、カードデッキを初期化してシャッフルし、5 枚のカードを配り、その結果を出力せよ。

《関数 `updateHand()`》

ライブラリ	<code>card.h</code>
用法	<code>Card *updateHand(Card *hand, int num, Card deal);</code>
処理内容	ポインタ <code>hand</code> が指す配列の <code>num</code> 個の要素と <code>deal</code> を格納する配列を新たに確保し、その配列の先頭アドレスを返す
実装方法	<ol style="list-style-type: none">1. <code>hand</code> が指す配列（動的に確保されている）の要素数 <code>num</code> より 1 つ大きい要素数をもつ Card 型配列を <code>calloc()</code> により確保2. <code>hand</code> が指す配列の内容と <code>deal</code> を新たに確保した配列にコピー3. <code>hand</code> が指す配列の領域を <code>free()</code> により消去4. 新しい配列へのポインタを返す

[ヒント 1] 要素数 `num` 個の `obj` 型 (`int`, `double`, `Card` など) 配列 `array` のメモリ領域を `calloc()` (`stdlib.h`) により確保する場合、次のように呼び出す。

```
array = calloc((size_t) num, (size_t) sizeof(obj));
```

例 1) 要素数 100 個の `double` 型配列の場合

```
array = calloc((size_t) 100, (size_t) sizeof(double));
```

例 2) 要素数 `N` 個の `int` 型配列の場合、

```
array = calloc((size_t) N, (size_t) sizeof(int));
```

注) `calloc()` と `malloc()` の違い

```
calloc(): ptr = calloc((size_t) count, (size_t) size);
```

`size` バイトの領域を連続して `count` 個確保し、そのアドレスを返す

```
malloc(): ptr = malloc((size_t) size);
```

`size` バイトの領域を確保して、そのアドレスを返す

[ヒント 2] ディールは、シャッフルされたカードデッキ `deck` の添字 `i` をインクリメントすることで実装できる。つまり、`deck[0]` がカードデッキの一番上のカード、`deck[1]` がその下のカード、・・・、と考えることができる。

[補足] ブラックジャックでは最終的な手札の枚数は不確定、つまり、スタンド/ヒットの選択により可変である。可変長のデータを扱う場合、リストによる実装が一般的である。上記実装方法以外にリストを用いて実装してもよい。しかしその場合、構造体定義を変更する必要があることに注意すること。

【課題 4-3】 手札からポイントを計算する関数 `calcPoint()` を作成して `card.c` に追加し、配られたカードのポイントを出力せよ。但し、フェースの J、Q、K は 10 としてカウントすること。

《関数 `calcPoint()`》

ライブラリ	<code>card.h</code>
用法	<code>int calcPoint(Card *hand, int num);</code>
処理内容	Card 型配列へのポインタ <code>hand</code> が指す配列に格納された <code>num</code> 枚のカードのポイントを計算して返す

[ヒント] フェースは文字列により表されていることに注意。つまり、“4”、“10”は整数の 4、10 ではない。文字列の“4”、“10”を整数に変換する関数として関数 `atoi()` (`stdlib.h`) がある。`hand[0].face` が“10”で、`spot` を `int` 型変数とするとき、

```
spot = atoi(hand[0].face);
```

と呼び出すと、`spot` には 10 が代入される。

第 5, 6 回 トランプゲームの作成

【課題 5-1】 第 2 回の課題で作成したサイコロによるブラックジャックをトランプによるブラックジャックに書き換えよ。

【make コマンドの利用】

規模の大きなプログラムを開発する場合、関連する処理ごとにファイルを分けてコーディングをすることにより、プログラム全体が整理され、開発の効率があがる場合がある。しかしその場合、複数のソースファイルの管理を正しく行う必要も生じる。UNIX 環境では、ソースファイルの管理をサポートするツールとして make コマンドが広く用いられている。

make コマンドは、編集済みのソースファイルから実行ファイル生成までの過程の処理を管理する。実際には、処理の内容を makefile と呼ばれるファイルに記述する。例えば、これまでに作成したソースプログラム main.c、card.c、ヘッダファイル card.h から実行ファイル blackjack を生成する場合、makefile には次のように記述する。

《makefile》

```
bjack: card.o main.o
    gcc card.o main.o -o blackjack
card.o: card.c
    gcc -c card.c
main.o: main.o
    gcc -c main.c
```

以上の makefile をソースファイルと同じディレクトリ保存し、同じディレクトリで make コマンドを入力すれば、

```
% make
gcc -c card.c
gcc -c main.c
gcc card.o main.o -o blackjack
%
```

とコンパイルを実行し、実行ファイル blackjack を生成する。

makefile に記述するのは、実行ファイル生成のためのルールであり、そのルールは、ターゲット (target)、必須項目 (prereqs)、実行コマンド (commands) の3つで次のように構成される。

```
target: prereq1 prereq2 . . .
    command1
    command2
    . . .
```

実行コマンド `commands` の前にはタブ文字を入力することに注意。

各ルールは、ターゲットを生成するための実行コマンドを記述している。ファイルの先頭からルールが適用されるが、ターゲット作成に必要な必須項目が存在しないと、その必須項目の生成に該当するターゲットを作成するルールが優先される。ルールを誤って記述してしまうと、プログラムを構築できない。ルールを正しく記述することで、複数ソースプログラム間の依存関係を管理することができる。

【課題 5-2】 課題 5-1 のプログラムに対する `makefile` を記述し、`make` によりプログラムの生成を行え。

【発展課題】 すでにプログラムを完成させることができれば、さらに次のように拡張してもよい。

- ・ A は 1 か 11 のどちらかプレイヤーの都合の良い扱いに出来る。
例えば、配られた 2 枚のカードが A と J なら、 $11+10=21$ 。
- ・ 1 ゲームだけでなく、ユーザが望む限り何ゲームでも続けて出来る。
- ・ プレーヤーを複数にする。
- ・ 点数をかけることができる。

など

プログラミング演習3 データベースのグループ開発



情報理工学部知能情報学科
北野勝則, 谷口忠大

1. 演習の目的

「データベースをグループで作ろう」

- プログラミング演習1～2の知識に基づき, 最低限「使える」基礎的なプログラムをグループで開発する.
- 第6回までに習ったプログラムの分割開発法に基づいてプログラム開発をグループで分担開発する.
- プログラムの設計・開発・テストの基本的なプロセスを体験する. グループ開発の困難さを理解し仕様の作成やコメント記述といったコミュニケーション技術を身につける.

2. 問題の場面設定

- 登場人物



- 上司

- あなたの勤務するソフトウェア開発代行会社K社でのあなたの上司。自分のやりたくない仕事は部下に丸投げする事で有名。



- 風呂苑社長

- M駅側の銭湯「風呂苑」の社長。上司と旧友でたまに、本当に必要なのかどうか分からないソフトウェアを発注してくる。



- あなた

- 中小企業K社でソフトウェア開発を行う駆け出しのSE。入社したときに面接で現上司に「C言語もままなりません！」と言ったのに採用された。上司に「心配するな、俺が鍛えてやる。」と言われ、内心びびっている。

ある日の会話（風呂苑社長と上司①）

- ある日、K社のオフィスでM駅側で銭湯「風呂苑」を営む社長が、あなたの上司と紅茶を飲みながら話しています。

社長「いやねえ、簡単なプログラムなんですよ。ただ単に人の名前とメールアドレスを管理するだけという～^^」

上司「ええっ？そんな物、いくらでもあるじゃないですか？」

社長「いや、その、古～いPCでね。Windowsも入ってないんですよ。全部コマンドラインでやってるPCで～。」

上司「はあ。しかし、なんでわざわざそんなPCを・・・」

社長「古参の銭湯業界には銭湯業界ならではの事情というのがあるんですよ！……まあ、深く詮索せんで下さい。」



ある日の会話(風呂苑社長と上司②)

上司「はあ、まあ、そんな難しくも無いですし、料金さえいただければ全然つくりますけどね。・・・あれ、じゃあMySQLとかのデータベースソフトも入らないのか!??」

社長「正直よくわかりませんが、C言語のプログラムのコンパイルは出来るらしいですよ。」

上司「ネ・・・ネイティブCですか!(ん?待てよ、例の新入社員にやらせる練習問題にちょうどいいかも..)わかりました。お引き受けします。」

社長「ありがたいわ。たすかるで～。機能は大体おまかせするわ」

上司「納期はどうしましょう?」

社長「早いほうがええけど、任すで。」

上司「・・・では、五日で仕上げましょう(ニヤリと不敵な笑み)」



□MySQL 有名なフリーのデータベース環境。最近のHPなどでのウェブサービスはMySQL+PHP, MySQL+Rubyなどで構築されることが多い。

□ネイティブC 単純なC言語しか使わずに組むプログラムの事。

ある日の会話(上司とあなた)

- ある日、出勤するとニヤニヤしながら、上司があなたを呼びました

上司「きみきみ、プログラミングの仕事がでてきたよ。」

あなた「ええ?C言語もろくに出来ないですよ。」

上司「いっただろ、『鍛えてやる』って。ちょうど、君と同じようなスタートラインにいるメンバーが2,3名いるからトレーニング兼ねて作るといい。」

あなた「・・・わかりました。いつまでですか?」

上司「5日後だ。」

あなた「えええええええ～??」

- というわけで、あなたはコマンドラインから名前とメールアドレスなどの入出力が可能なデータベースを作る事になりました。もう少し、具体的な要求仕様は以下です。



3. 要求仕様の概略

必要条件

□ メールアドレス管理データベース

1. 最低限, 人の名前(ローマ字)とメールアドレスを管理できる. データ数に上限を設けない.
2. 一連の入力, 編集, 出力, 消去などの作業(CRUD操作)が出来る.
3. グループを設定できる.
4. グループ毎に表示できる.
5. 名前をA~Z順に並べ替えて表示できる.
6. 名前で検索できる(部分一致).
7. CSVファイルにデータを出力・入力できる.
8. その他の追加機能(※オプション課題)

CRUD操作

- CRUD とは, データベースを始めほとんど全てのコンピュータソフトウェアが持つべき4つの基本機能のイニシャルを並べた用語. Create(生成)、Read(読み取り)、Update(更新)、Delete(削除)の頭文字である.
 - Create 新しいデータの入力・追加
 - Read 入力済みのデータの検索・表示
 - Update 入力済みのデータの編集・更新
 - Delete 入力済みのデータの消去

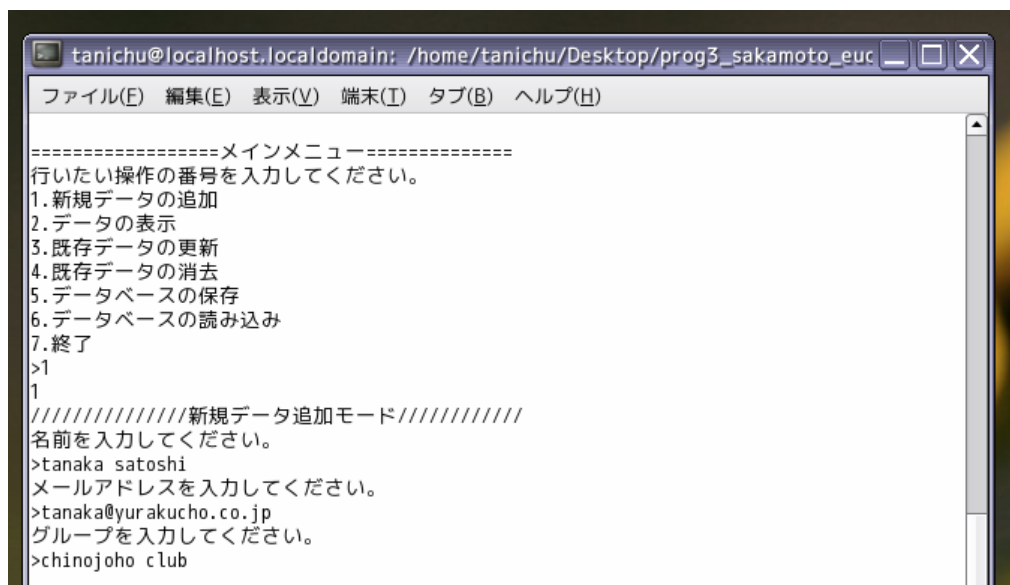
CSVファイル

- Comma-Separated Values (CSV) ファイルは、いくつかのフィールド(項目)をコンマ「,」で区切ったテキストファイル
- メーカーや住所録を始め多くのデータベースで入出力可能な形式
- WindowsならダブルクリックでExcelで開けます。

【CSVファイルの一

```
例) #1,Name,Mail Address,Group  
2,tanaka satoshi,tanaka@yurakucho.co.jp,chinojoho club  
3,sugita toru,toru@komatsuna.jp,chinojoho club  
4,teito mamoru,teito@ubi.org,inakagurashi  
5,keita ohtori,keita@himaraya.jp,none
```

コマンドラインで入出力 ～対話型インタフェース～



```
tanichu@localhost.localdomain: /home/tanichu/Desktop/prog3_sakamoto_euc  
ファイル(E) 編集(E) 表示(V) 端末(I) タブ(B) ヘルプ(H)  
=====メインメニュー=====  
行いたい操作の番号を入力してください。  
1. 新規データの追加  
2. データの表示  
3. 既存データの更新  
4. 既存データの消去  
5. データベースの保存  
6. データベースの読み込み  
7. 終了  
>1  
1  
1  
////////////////////新規データ追加モード////////////////////  
名前を入力してください。  
>tanaka satoshi  
メールアドレスを入力してください。  
>tanaka@yurakucho.co.jp  
グループを入力してください。  
>chinojoho club
```

※ 必ずしも厳密にこのようになる必要はないが、コマンド入力で分岐して、入出力作業を行う事が出来るように作る。

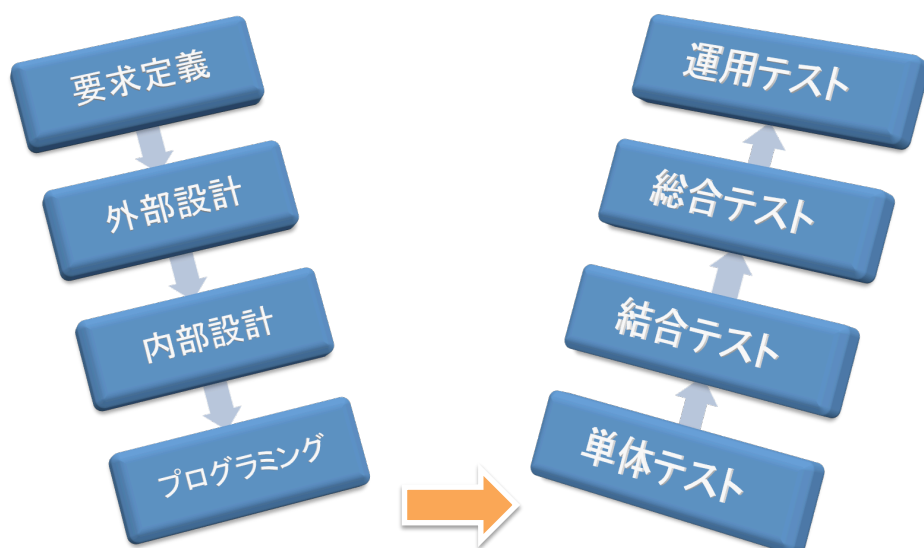
その他の指定

1. 1件のデータは構造体として管理して下さい.
2. データ入力の度にメモリ確保を行い, データ構造はリストとして作って下さい.
3. 名前は同姓同名が存在する可能性があるの
ので, 名前とは別にユニークIDを作ること.

※ ユニークID
データベースのデータを一意に決めることの出来るID

4. 開発のプロセスと課題 「ウォーターフォールモデル」

- 最も単純なシステム開発のプロセス



大まかにこのような流れに従って開発をすすめましょう.

開発のプロセスと課題

- 第一週(第7回)
 - 課題内容と開発概略の説明
 - グループとリーダーを決める
 - 作成機能の整理
 - どのような機能を持つプログラムを作るのか？
オプション課題の追加機能を含め一覧表を作成する.
 - ✓機能一覧表の作成 => 課題チェック [cp 1-1]
 - 外部設計
 - 設計した機能をどのようなインタフェース(ユーザの入力と画面表示)で提供しユーザをナビゲートするのか？画面遷移図を作成する.
 - ✓画面遷移図の作成 => 課題チェック [cp 1-2]

開発のプロセスと課題

- 第二週(第8回)
 - 内部設計
 - メールデータベースのデータ構造はどのようにするのか
=>変数宣言の決定
 - 全ての機能とインタフェースを実現するためにどのような関数を作成し、どのようにソースコードを分割するのか？
=> 作る関数の種類の決定と開発の割り振りの決定.
 - ✓関数毎に開発責任者の名前をコメントで記入した
ヘッダファイルの作成 => 課題チェック[cp 2]
 - 終わった人はプログラミングに入って下さい.

開発のプロセスと課題

• 第三週～第四週(第9,10回)(個人作業)

– プログラミング

- 自分の担当になった関数を作成する.
- 出来る限り第四週までに作ってくるようにしましょう.

– 単体テスト

- 自分の担当になった関数がちゃんと作動するかどうかを, その関数を利用したプログラムを作って検証する.

✓関数の単体テスト => 課題チェック[cp 4]



開発のプロセスと課題

• 第五週(第11回)

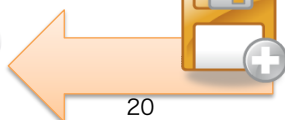
– 結合テスト・総合テスト

- 完成した部品をmain関数の中に組み込んでいきます.
- その上で, あらかじめ設定されたテスト課題が実行できるか点検して下さい.
- エラーが生じる場合は適宜原因を見つけだし, デバッグしてください.

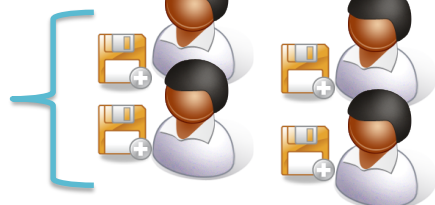
✓総合テスト(動作確認) => 課題チェック[cp 5]

– 運用テスト

- 実際に顧客先で運用し, 正しく動作するかをテストします. これは今回の演習では行いません.



20



[参考資料] 機能一覧表の例

※基本的に自由に書いて下さい。

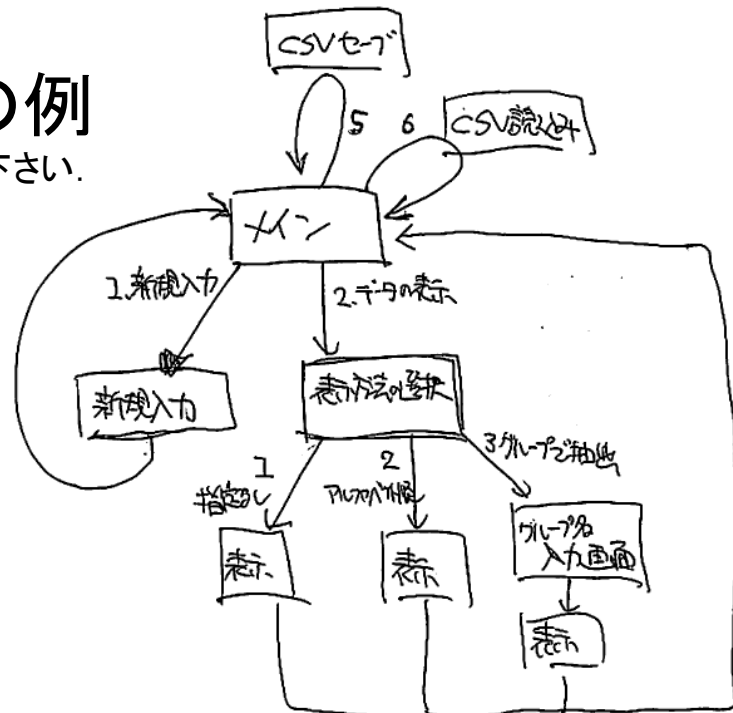
✕機能一覧表

1. リストのデータを順に表示する。
2. リストのデータを名前順にソートして表示する。
3. 入力された文字列に対し、グループ名と部分一致するデータだけをリストから出力する。
4. 指定したIDのデータに新しい値を代入する。



[参考資料] 画面遷移図の例

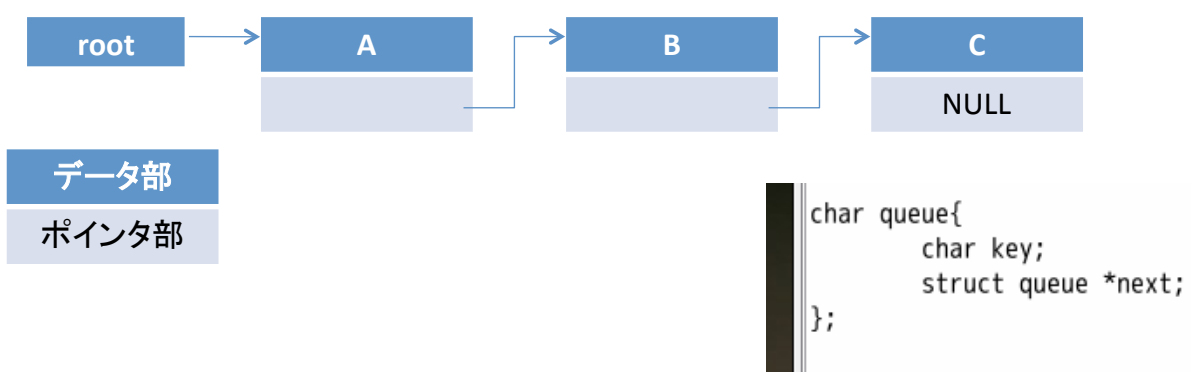
※基本的に自由に書いて下さい。



※ これはあくまで一例です。
自分たちが理解できるように書いて下さい。

[参考資料] リスト構造

- リストとはデータが次のデータの位置に関する情報を持っているデータ構造.
- 「データ構造とアルゴリズム」及び「プログラミング演習2」の復習をしましょう.



この演習に「答え」はありません。
グループ+TAで相談し、風呂苑社長の要望を満たすプログラムを開発して下さい。

第 12～15 回 自由課題およびレポートについて

◎ 課題

これまで学んだCプログラミングの知識を用いて、1つプログラムを作成せよ。但し、インターフェースはテキストベースでよいものとする。

例) ゲーム、実用的なユーティリティ、など

◎ レポート

自由課題で作成したプログラムに関するレポートを作成せよ。次の項目を必須とする。

1. なぜそのプログラムを課題として選んだか
2. 処理手順の説明
3. ソースプログラム
4. 適当な入力に対する実行結果
5. 作成したプログラムに対する考察

表紙に、科目名、クラス、学籍番号、名前も忘れないこと。

◎ 評価

- ・ 期限までに提出されたか（提出方法については追って連絡予定）
- ・ 上記必須項目が含まれているか
- ・ 作成したプログラムは正しく動くか
- ・ アイデアに独創性はあるか