

1. 問題設定

サイコロを 100 回振って出目の平均を求めるプログラムを作成する

2. 問題分析

今回の問題について、以下の点を特に注意することにする

- (概要) 生成された乱数を処理して、その結果を出力する
- printf 関数で出力する (C 言語には print 関数が存在しない)
- printf 関数の第 1 引数に、任意の書式指定子を含む文字列リテラルを与える
- printf 関数の第 2 引数以降には、書式指定子に代入する変数を順に入力する
- printf 関数を使用するため、"stdio.h" を include する
- rand 関数および、srand 関数を使用するため、"stdlib.h" を include する
- time 関数を使用するため、"time.h" を include する
- Linux の場合は、usleep 関数を使用するため、"unistd.h" を include する
- Windows の場合は、sleep 関数を使用するため、"windows.h" を include する
- スタート関数 (main 関数) の戻り値には 0 を指定する (この値が終了コードとなる)
- 文字列の改行には、エスケープ文字の "\n" を用いる
- 余りの計算には、剰余演算子 (%) を用いる
- 平均の計算には、double 型 (倍精度浮動小数点型) を使用する
- 今回のレポートは、プログラムの演習も兼ねて、PDF 内で端末の一部機能を再現する (実装コマンドについては後に記述)

3. 設計

この問題で作成するプログラムを以下のような流れ図（フローチャート）で示す

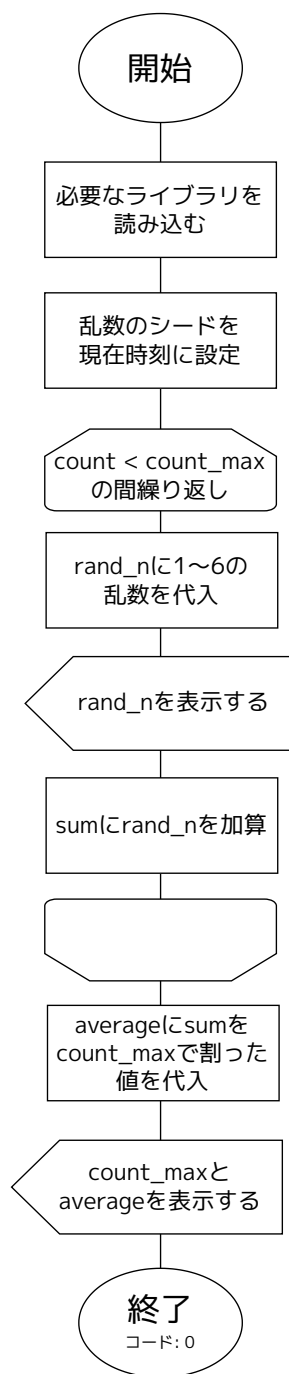


図 1: 条件分岐と繰り返し（フローチャート）

4. 実装1

ソースコード 1: prog01.c

```
/*
サイコロを 100回振って出目の平均を求める
*/

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

int main(void) {
    int count;
    int count_max = 100;
    int rand_n;
    double sum = 0.0;
    double average;

    srand((unsigned int)(time(NULL)));

    for (count = 0; count < count_max; count++) {
        rand_n = (rand() % 6) + 1;
        printf("%d \n", rand_n);
        sum += rand_n;
    }

    average = sum / (count_max);
    printf("%d 回の平均値は %.2f \n", count_max, average);

    return 0;
}
```

プログラムの解説

- 5 行目 `stdio.h` を include する
- 6 行目 `stdlib.h` を include する
- 7 行目 `time.h` を include する
- 9 行目 `main` 関数を定義する
- 10 行目 変数 `count` を整数型として宣言する
- 11 行目 変数 `count_max` を整数型として宣言する
- 12 行目 変数 `rand_n` を整数型として宣言する
- 13 行目 変数 `sum` を倍精度浮動小数点型として宣言する
- 14 行目 変数 `average` を倍精度浮動小数点型として宣言する
- 16 行目 `srand` 関数で、乱数のシードを現在時刻に設定する
- 18 行目 `for` 文で `count_max` 回だけ繰り返す
- 19 行目 `rand_n` に `rand` 関数の値を 6 で割った余りに 1 加算した結果を代入する
- 20 行目 `printf` 関数で `rand_n` の値を出力する
- 21 行目 `sum` に `rand_n` を加算する
- 24 行目 `average` に `sum` を `count_max` で割った結果を代入
- 27 行目 `main` 関数の戻り値として 0 を返す

用途	説明	変数名	データ型
演算	ループ回数	<code>count</code>	<code>int</code>
演算	ループ最大回数	<code>count_max</code>	<code>int</code>
演算	乱数格納	<code>rand_n</code>	<code>int</code>
演算	乱数の合計	<code>sum</code>	<code>double</code>
演算	乱数の平均	<code>average</code>	<code>double</code>

図 2: 変数表 (prog01.c)

5. 検証 1

2 ページ目の PDF コンソールにて以下のコマンドで実行可能です

`./prog01.out`

```
1
2
6
5
2
4
3
1
1
6
3
1
5
6
2
4
5
3
3
5
4
2
2
2
4
6
1
1
2
5
3
2
6
3
1
4
5
5
100 回の平均値は 3.29
s24138@ubuntu-pdf:~$
```

図 3: コンソール 1 (prog01.c)

6. 実装2

ソースコード 2: prog02.c

```
/*
サイコロを振り続けて出目の出現数と平均を求める
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h> // UNIX
// #include <windows.h> // Windows

#define DICE 6
#define GRAPH_W 100
#define GRAPH_S 100

int length(int *array) {
    return DICE;
}

void drawGraph(int *array) {
    int max = 0;

    for(int i = 0; i < length(array); i++) {
        if(array[i] > max) max = array[i];
    }

    double times = 1;

    if(max/GRAPH_S > GRAPH_W) times = ((double) GRAPH_W / ((double) max * (double)
        GRAPH_S));

    for(int i = 0; i < length(array); i++) {
        printf("[%d] ", i + 1);
        for(int j = 0; j < array[i]*times/((double) GRAPH_S); j++) {
            printf("=");
        }
        printf(" %d\n", array[i]);
    }
}

double getAverage(int *array) {
    int sum = 0;
    int count = 0;
    for(int i = 0; i < length(array); i++) {
        sum += array[i] * (i + 1);
        count += array[i];
    }

    return (double) sum / (double) count;
}

int getCount(int *array) {
    int count = 0;
    for(int i = 0; i < length(array); i++) {
        count += array[i];
    }
    return count;
}
```

```

int main(void) {
    int count = 0;
    int count_one_loop_max = 5;
    int rand_n;
    int sum = 0;
    double average;
    int count_array[DICE] = {0};

    srand((unsigned int)(time(NULL)));

    while (1) {
        for (int i = 0; i < count_one_loop_max; i++) {
            rand_n = (rand() % DICE) + 1;
            count_array[rand_n - 1]++;
        }

        system("clear");
        drawGraph(count_array);
        average = getAverage(count_array);
        count = getCount(count_array);
        printf("Average: %.2f , Count: %d\n", average, count);
        usleep(10);
    }

    return 0;
}

```

用途	説明	変数名	データ型
演算	ループ回数	count	int
演算	ループ最大回数	count_max	int
演算	乱数格納	rand_n	int
演算	乱数の合計	sum	double
演算	乱数の平均	average	double
演算	各出目の回数	count_array	int *[]

図 4: 変数表 (prog02.c)

7. 検証2

2 ページ目の PDF コンソールにて以下のコマンドで実行可能です

`./prog02.out`

```
[1]===== 1219
[2]===== 1239
[3]===== 1266
[4]===== 1215
[5]===== 1196
[6]===== 1180
Average: 3.47 , Count: 7315
```

図 5: コンソール 2 (prog02.c)

8. 補足

以下は、今回の PDF コンソールのソースコードである

Github リポジトリ—<https://github.com/mumu17-git/PDFConsole>

ソースコード 3: pdfconsole.js

```
//-----prog01/02-----

const prog01_c = {type: "c", content:
、
/*
サイコロを 100回振って出目の平均を求める
*/

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

int main(void) {
    int count;
    int count_max = 100;
    int rand_n;
    double sum = 0.0;
    double average;

    srand((unsigned int)(time(NULL)));

    for (count = 0; count < count_max; count++) {
        rand_n = (rand() % 6) + 1;
        printf("%d \\n", rand_n);
        sum += rand_n;
    }
}
```



```

    }

    average = sum / (count_max);
    printf("%d 回の平均値は %.2f \n", count_max, average);

    return 0;
}
~
};

function prog01_out_js() {
    var outs = "";
    var count = 0;
    var rand_n = 0;
    var count_max = 100;
    var sum = 0;
    var average = 0.0;
    for(count = 0; count < count_max; count++) {
        rand_n = rand(6)+1;
        outs += `${rand_n} \n`;
        sum += rand_n;
    }

    average = sum / count_max;
    outs += `${count_max} 回の平均値は ${average} \n`;

    function rand(max) {
        return Math.floor(Math.random() * max);
    }

    return outs;
}

let prog01_out = {type: "out", timing: -1, content: prog01_out_js()};

const prog02_c = {type: "c", content:`
/*
サイコロを振り続けて出目の出現数と平均を求める
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h> // UNIX

```

```

//#include <windows.h> // Windows

#define DICE 6
#define GRAPH_W 100
#define GRAPH_S 100

int length(int *array) {
    return DICE;
}

void drawGraph(int *array) {
    int max = 0;

    for(int i = 0; i < length(array); i++) {
        if(array[i] > max) max = array[i];
    }

    double times = 1;

    if(max/GRAPH_S > GRAPH_W) times = ((double) GRAPH_W/(double) max * (double)
    GRAPH_S);

    for(int i = 0; i < length(array); i++) {
        printf("[%d] ", i + 1);
        for(int j = 0; j < array[i]*times/(double) GRAPH_S; j++) {
            printf("=");
        }
        printf("    %d\n", array[i]);
    }
}

double getAverage(int *array) {
    int sum = 0;
    int count = 0;
    for(int i = 0; i < length(array); i++) {
        sum += array[i] * (i + 1);
        count += array[i];
    }

    return (double) sum / (double) count;
}

int getCount(int *array) {
    int count = 0;
    for(int i = 0; i < length(array); i++) {
        count += array[i];
    }
}

```

```

        return count;
    }

int main(void) {
    int count = 0;
    int count_one_loop_max = 5;
    int rand_n;
    int sum = 0;
    double average;
    int count_array[DICE] = {0};

    srand((unsigned int)(time(NULL)));

    while (1) {
        for (int i = 0; i < count_one_loop_max; i++) {
            rand_n = (rand() % DICE) + 1;
            count_array[rand_n - 1]++;
        }

        system("clear");
        drawGraph(count_array);
        average = getAverage(count_array);
        count = getCount(count_array);
        printf("Average:  %.2f , Count: %d\n", average, count);
        usleep(10);
    }

    return 0;
}
-
};

const DICE = 6;

let prog02_out_js_variable = {
    count: 0,
    count_one_loop_max: 5,
    rand_n: 0,
    sum: 0,
    average: 0.0,
    count_array: Array(DICE).fill(0)
}

function prog02_out_js() {
    const GRAPH_W = 90.0;
    const GRAPH_S = 5.0;

```

```

function drawGraph(array = []) {
    var max = 0.0;
    for(var i = 0;i < array.length;i++) {
        if(array[i] > max) max = array[i];
    }

    var times = 1;

    if(max/GRAPH_S > GRAPH_W) times = (GRAPH_W/max*GRAPH_S);

    d_clear();

    for(var i = 0;i < array.length;i++) {
        d_line(`[${i+1}]`, false);
        for(var j = 0;j < array[i]*times/GRAPH_S;j++) {
            d_line("=", false);
        }
        d_line(`  ${array[i]}`);
    }
}

function getAverage(array = []) {
    var sum = 0;
    var count = 0;
    for(var i = 0;i < array.length;i++) {
        sum += array[i] * (i + 1);
        count += array[i];
    }

    return sum / count;
}

function getCount(array = []) {
    var count = 0;
    for(var i = 0;i < array.length;i++) {
        count += array[i];
    }

    return count;
}

function rand(max) {
    return Math.floor(Math.random() * max);
}

function loop() {
    for(var i = 0;i < prog02_out_js_variable.count_one_loop_max;i++) {

```

```

        prog02_out_js_variable.rand_n = rand(DICE) + 1;
        prog02_out_js_variable.count_array[prog02_out_js_variable.rand_n - 1]++;
    }

    drawGraph(prog02_out_js_variable.count_array);
    prog02_out_js_variable.average = getAverage(prog02_out_js_variable.count_array);
    prog02_out_js_variable.count = getCount(prog02_out_js_variable.count_array);
    d_line(`Average:  ${prog02_out_js_variable.average.toFixed(2)} , Count: ${
    prog02_out_js_variable.count}`);
}

function main() {
    while(whileProgLooping) {
        loop();
    }
}

loop();
}

let prog02_out = {type: "out", timing: 0, content: prog02_out_js};

//----- end -----

function init() {
    if (global.initialized) return;
    global.initialized = true;

    global.count = 1;

    cinput.value = "";
    consoleContents = currentCmdString;

    setDirContents(currentPath_List);

    setDirContents(currentPath_List, {'prog01.c': prog01_c, 'prog01.out': prog01_out
    });
    setDirContents(currentPath_List, {'prog02.c': prog02_c, 'prog02.out': prog02_out
    });

    countdown();
}

function onCfieldInput() {
    var iscp = isCurrentCmdString();

```

```

    if(iscp == 0) {
        consoleContents = consoleContents + event.change;
    }
}

function draw() {
    drawConsole();
}

var cinput = this.getField('console_input');
var cdisplay = this.getField('console_display');

var isLogShowing = false;
var whileProgLooping = false;
function drawConsole() {
    var iscp = isEntered();
    if(iscp == 0) {

    }else if(iscp == 1) {
        var cmd = cinput.value.toString();
        consoleContents += cmd+"\n";
        var cmd_split = cmd.split(" ");
        runFunction(cmd_split);
        cinput.value = "";
        consoleContents += currentCmdString;
    } else {
        cinput.value = "";
    }
    if(!isLogShowing&&!whileProgLooping)
        cdisplay.value = consoleContents;
}

function logSplit() {
    const line_max = 45;
    var cc_split = consoleContents.split("\n");
    if(cc_split.length > line_max) {
        for(var i = 0; i < Math.floor(cc_split.length / line_max); i++) {
            var c_array0 = cc_split.slice(line_max*i, line_max*(i+1)-1);
            consoleContents_history.push(c_array0.join("\n"));
        }
        var c_array1 = cc_split.slice(-(cc_split.length % line_max));
        consoleContents = c_array1.join("\n");
    }
}

function runFunction(cmd = []) {
    let str = cmd[0];

```

```

if(str.includes("."+getFilePath_Separator())) {
    str = "."+getFilePath_Separator();
    cmd[1] = cmd[0].replace("."+getFilePath_Separator(),"");
}
switch(str) {
    case "pwd":
        c_pwd();
        break;
    case "ls":
        c_ls();
        break;
    case "mkdir":
        var arg = "";
        if(cmd.length > 1) arg = cmd[1];
        c_mkdir(arg);
        break;
    case "."+getFilePath_Separator():
        c_crtDirCnt(cmd[1]);
        break;
    case "cat":
        c_cat(cmd[1]);
        break;
    case "log":
        c_log(Number(cmd[1]));
        break;
    case "exit":
        c_exit();
        break;
    default:
        o_line("Command not found");
        break;
}
}

function isEntered() {
    var v = cinput.value;

    if(v.length >= 1) {
        return 1;
    }

    return 0;
}

//0: 入力中, 1: 実行待, 2: 入力エラー
function isCurrentCmdString() {

```

```

var v = cinput.value;
if(v.length < currentCmdString.length)
    return 2;
if(v.lastIndexOf(getCmdSign()) == -1)
    return 2;
if(v.slice(0, v.lastIndexOf(getCmdSign())+2) !== currentCmdString) {
    consoleContents += v.slice(0, v.lastIndexOf(getCmdSign())+2);
    return 2;
}
if(currentCmdString.toString() !== v.toString())
    return 1;
return 0;
}

function getCmd_String_Default() {
    return getUser()+"@"+getComputerName()+":"+getPath_String_Current()+getCmdSign
        ()+" ";
}

function getCmdSign() {
    return "$";
}

function getFilePath_Separator() {
    return "/";
}

function getUser() {
    return "s24138";
}

function getComputerName() {
    return "ubuntu-pdf";
}

function getPath_String_Current() {
    var str = currentPath_List.join(getFilePath_Separator());
    if(currentPath_List.length <= 1)
        str = getFilePath_Separator();
    var home_dir = getPath_List_Home().join(getFilePath_Separator());
    if(str.includes(home_dir))
        str = str.replace(home_dir, getPath_Sign_Home());

    return str;
}

function getPath_Sign_Home() {

```



```

    return "~";
}

function getPath_List_Home() {
    return ["", "home", getUser()];
}

function setDirContents(path = [], contents = {}) {
    getDirContents(path);
    var obj = mergeDeeply(getDirContents_tmp0, contents, {concatArray: true});
    const tramped = trampoline(createDirContents);
    tramped(path, dirContents, obj);
}

let getDirContents_tmp0 = {};
function getDirContents(path = []) {
    getDirContents_tmp0 = {};
    const tramped = trampoline(gettingDirContents);
    tramped(path, dirContents);
}

const createDirContents = (path = [], originObj = {}, obj = {}, contentObj = {},
    idx = -2) => {
    if(idx == -2) idx = path.length - 1;
    if(path.length <= 0 || idx < 0) {
        dirContents = mergeDeeply(originObj, obj, {concatArray: true});
        return true;
    }
    if(!Object.keys(obj).includes(path[idx]))
        var obj_t = contentObj;
        obj_t[path[idx]] = obj;
    idx--;
    path = path.slice(0, -1);
    getDirContents(path);
    return () => createDirContents(path, originObj, obj_t, {}, idx);
}

const gettingDirContents = (path = [], obj = {}, idx = -2) => {
    if(idx == -2) idx = 0;
    if(path.length <= 0 || idx > path.length-1) {
        getDirContents_tmp0 = Object.assign({}, obj);
        return true;
    }
    if(Object.keys(obj).includes(path[idx]))
        var obj_t = Object.assign({}, obj[path[idx]]);
    idx++;

```

```

    return () => gettingDirContents(path, obj_t, idx);
}

function trampoline (fn) {
  return (...args) => {
    let result = fn(...args);
    while (typeof result === 'function') {
      result = result();
    }
    return result;
  };
}

function mergeDeeply(target, source, opts) {
  const isObject = obj => obj && typeof obj === 'object' && !Array.isArray(obj);
  const isConcatArray = opts && opts.concatArray;
  let result = Object.assign({}, target);
  if (isObject(target) && isObject(source)) {
    for (const [sourceKey, sourceValue] of Object.entries(source)) {
      const targetValue = target[sourceKey];
      if (isConcatArray && Array.isArray(sourceValue) && Array.isArray(
targetValue)) {
        result[sourceKey] = targetValue.concat(...sourceValue);
      }
      else if (isObject(sourceValue) && target.hasOwnProperty(sourceKey)) {
        result[sourceKey] = mergeDeeply(targetValue, sourceValue, opts);
      }
      else {
        Object.assign(result, {[sourceKey]: sourceValue});
      }
    }
  }
  return result;
}

var currentPath_List = getPath_List_Home();
var currentCmdString = getCmd_String_Default();
var consoleContents;
var consoleContents_history = [];

var dirContents = {};

function wrappedDraw() {
  try {
    draw();
  }
}

```

```

    } catch (e) {
        app.alert(e.toString())
    }
}

function start() {
    app.setInterval('wrappedDraw()', 15);
}

function countdown() {

    global.count--;
    if (global.count < 0) {
        start();
    } else {
        app.setTimeout('countdown()', 1000);
    }
}

init();

function bash_update(name = "", type = "") {
    switch (name) {
        case "prog01.out":
            prog01_out = {type: type, content: prog01_out_js()};
            setDirContents(currentPath_List, {'prog01.out': prog01_out});
            break;
    }
}

function o_line(text = "") {
    consoleContents += text+"\n";
    logSplit();
}

function d_line(text = "", lb = true) {
    cdisplay.value = cdisplay.value+text;
    if(lb) cdisplay.value = cdisplay.value+"\n";
}

function d_clear() {
    cdisplay.value = "";
}

function c_pwd() {
    var str = currentPath_List.join(getFilePath_Separator());
    if(currentPath_List.length <= 1)

```

```

    str = getFilePath_Separator();

    o_line(str);
}

function c_ls() {
    if(currentPath_List.length < 1) {
        o_line(Object.keys(dirContents).join("    "));
        return;
    }

    getDirContents(currentPath_List);
    if(Object.keys(getDirContents_tmp0).length > 0)
        o_line(Object.keys(getDirContents_tmp0).join("    "));
}

function c_mkdir(name = "") {
    if(name== "") {
        o_line("mkdir: missing operand");
        return;
    }

    setDirContents(currentPath_List.concat([name]));
}

let timing0_interval = null;
function c_crtDirCnt(name = "") {
    if(name == "") {
        o_line("-bash: ./: Is a directory");
        return;
    }

    getDirContents(currentPath_List);

    if(!Object.keys(getDirContents_tmp0).includes(name)) {
        o_line(`-bash: ./${name}: No such file or directory`);
        return;
    }

    if (getDirContents_tmp0[name].type === "out") {
        if(getDirContents_tmp0[name].timing == -1) {
            o_line(getDirContents_tmp0[name].content);
            bash_update(name,getDirContents_tmp0[name].type);
        }else if (getDirContents_tmp0[name].timing == 0) {
            whileProgLooping = true;
            timing0_interval = app.setInterval("prog02_out_js()", 50);

```

```

    }

    }else {
        o_line("Permission denied");
    }
}

function c_cat(name = "") {
    if(name == "") {
        o_line("cat: ./: Is a directory");
        return;
    }

    getDirContents(currentPath_List);

    if(!Object.keys(getDirContents_tmp0).includes(name)) {
        o_line(`cat: ./${name}: No such file or directory`);
        return;
    }

    if (getDirContents_tmp0[name].type === "c") {
        o_line(getDirContents_tmp0[name].content);
    }else {
        o_line("Permission denied");
    }
}

function c_log(page) {
    if(isNaN(page)) {
        if(consoleContents_history.length > 1)
            o_line(`Input 0-${consoleContents_history.length-1} into the argument`);
        else if(consoleContents_history.length > 0)
            o_line("Input 0 into the argument");
        else
            o_line("Failed to load log");
        return;
    }

    if(consoleContents_history.length < page) {
        o_line("Failed to load resource");
        return;
    }else if(consoleContents_history.length == page) {
        c_exit();
        return;
    }
}

```

```

}

isLogShowing = true;
cdisplay.value = consoleContents_history.slice(page)[0];
}

function c_exit() {
  cdisplay.value = "";
  isLogShowing = false;
  whileProgLooping = false;
  app.clearInterval(timing0_interval);
}

```

以下は、今回の PDF コンソールで使用可能なコマンド一覧である

ls カレントディレクトリの中身を取得

pwd カレントディレクトリの絶対パスを表示

mkdir NAME カレントディレクトリに新規ディレクトリを追加する。NAME にはディレクトリ名を入力

./NAME カレントディレクトリにあるファイルを実行する。NAME にはファイル名を入力。タイプが out ファイルのみ

cat NAME カレントディレクトリにあるファイルのテキストを表示する。NAME にはファイル名を入力。タイプが c のファイルのみ

log PAGE コンソールの表示行数制限を超過した際に過去のコンソール履歴を閲覧できる。PAGE にはインデックスを入力。負の数を指定すると、最後のページからの探索される

exit log コマンドやファイルを実行を終了する

