計算機とアルゴリズム I

$3^4 \cdot 5^2$ 年度用

第 1 版

目次

第 1 章

まず始めに

1.1　計算機とアルゴリズム I

いろいろな問題 (数学や情報のみならず、日常生活等の問題) を考える場合、アルゴリズムを考えることが必須である。また、コンピュータを用いて問題を解くとき、プログラミングを行うことが必須条件となる。

この講義では、プログラミング言語の 1 つである C 言語について学習し、実際にコンピュータを利用した演習を行う。

最初は、思った通り動作するプログラムを作成することを目的とする。多少遠まわりをしたプログラミングでも自分で考え作成したものなら十分成果となる。

したがって、課題は全て完成していなくても、途中までででも、提出すること。その後、プログラムの効率化や人に説明しやすいプログラミングを学習する。

この講義では、各回の前半に講義を行い、後半を実習とする。

実習課題の提出はメールで行ってもらうので、パソコンでメールが送ることが出来るようにしておくことが必要である。その際、件名 (タイトル) が重要で、必ず 学生番号 課題番号 のみを書くようにすること。

例 S24M000 課題 1.1.3

学生番号と課題番号はドットも含めて半角で書くこと。

1.1.1　アルゴリズムとは

アルゴリズムとは、例えば料理の手順や、数学の証明のように、何か課題が与えられたとき、解決までの道筋 (処理手順) である。その際、たった一つのアルゴリズムだけで問題が解決することは少なく、また問題解決のためのアルゴリズムは一通りだけではない。

複数のアルゴリズムを組み合わせたり、一部を改良しながら作成し、問題解決を行うことが多い。

例 1.1.1. (カレー作り)

例えば、カレーを作るという課題があるとき、材料 (プログラムなら”データ (構造)”というもの) をレシピ (アルゴリズム) にそって処理して行く。

データ：

豚こま肉、じゃがいも、にんじん、たまねぎ、水、サラダ油、カレールー

アルゴリズム：

1. 洗ったじゃがいも、にんじんの皮をむき、洗う。
2. たまねぎは包丁で頭と根の部分を切り落とし皮をむく。
3. じゃがいも、にんじんを一口大に切る。
4. たまねぎを薄切りにする。
5. 厚手の鍋にサラダ油を熱し、豚こま肉と調理したじゃがいも、にんじん、たまねぎを入れ焦がさないように炒める。
6. 水を加え、沸騰させる。
7. あくを取ったら、中火で 20 分程度煮込む。
8. 火を止めて、カレールーを溶かしながら入れる。
9. 再び火をつけ、弱火でとろみがつくまで煮込む。
10. 完成

例 1.1.2. (数学の証明問題 [直接証明])

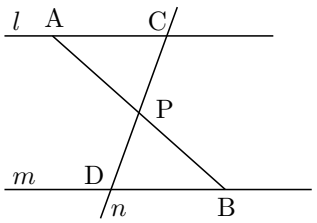
数学の証明問題 [直接証明] では、問い (の中にあるデータ) をもとに、文字式で表現したり、式変形して結論まで導く。

データ：

2 つの平行な直線 l, m がある。直線 l 上に点 A を、直線 m 上に点 B を取り、線分 AB の中点を P とする。点 P を通る直線 n と直線 l, m との交点を C, D とする。このとき、三角形 APC と BPD は合同であることを示せ。

アルゴリズム：

1. 仮定より AP=PB
2. 平行線の錯角より ∠PAC= ∠PBD
3. 対頂角より ∠APC= ∠BPD
4. 1 辺とその両端の角がそれぞれ等しい
5. 2 つの三角形の合同条件の 1 つを満たす
6. よって △APC= △BPD
7. 証明終了



♠ **補足**　例 1.1.1 の場合、手順は一通りではない。例えば、3. と 4. を入れ替えることこもできる。しかし、この例では手順の入れ替えは出来ない。

例 **1.1.3.** (ユークリッドの互除法)

C 言語のプログラムで、2 つの自然数 (データ a, b) の最大公約数を求める処理をして行く。

データ：

自然数 a, b の最大公約数を求める。

アルゴリズム：

1. $a < b$ の場合、a と b の値を入れ替える。
2. a を b で割った余りを r とする。
3. r が 0 でなければ a に b を代入し、b に r を代入して 2. に戻る。
4. r が 0 なら b が最大公約数となる。
5. 完成

実際の C 言語のプログラムの例：

```
#include <stdio.h>
int main(void){
   int a=35, b=25, r, tmp;
   if(a<b){
      tmp = a;
      a = b;
      b = tmp;
   }
   r = a % b;
   while(r!=0){
      a = b;
      b = r;
      r = a % b;
   }
   printf("最大公約数 = %d¥n", b);
   return 0;
}
```

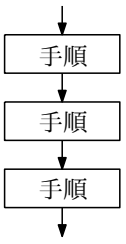
1.1.2 アルゴリズムの基本構造

アルゴリズムの基本構造は、順次 (逐次) 構造、選択構造、反復構造の 3 つであり、プログラムをこれらを組み合わせて作り上げていく。

★ 順次構造とは

”定められた順に、それぞれの処理を順番に 1 つずつ行う”

という構造である。プログラミングでは基本的に上の行から下の行まで順番に処理されていく。

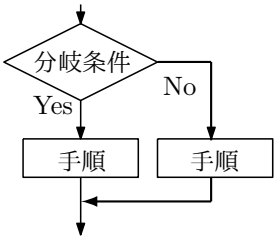


先にあげた例 1.1.2 の数学の証明問題はこの順次構造で処理を行っている。

★ 選択構造は、”ある条件によっていくつかの処理のうちどれか 1 つを選び、処理する”という構造である。

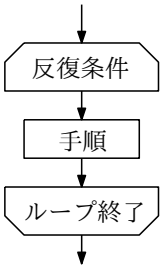
これは、プログラムの途中で 2 分岐, 3 分岐,... が生じる構造である。C 言語では if 文や switch 文が該当する。

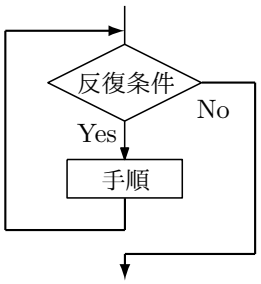
例 1.1.3 において、4 行目が (a<b が真か偽かによって処理を選ぶ) 分岐に相当する。



★ 反復構造は、”ある条件を満たすまで繰り返し処理を実行し、条件を満たしたら処理を終了する”という構造である。C 言語では for 文や while 文が該当する。

反復構造においては、右のフローチャートのように、反復条件を満たしている場合は、制御する”手順”を繰り返して処理する。

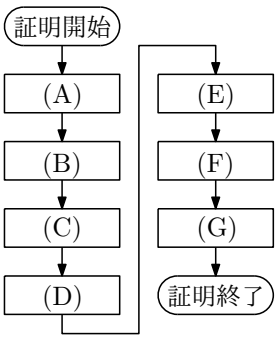




例 1.1.4. (**数学証明問題**)

連続する奇数の積に 1 を加えた数を N とすると、N は 4 の倍数になることを示せ。(解答は 1 通りではない)

(1) N= 奇数 × 奇数 +1
(2) 連続する奇数は $2n-1, 2n+1$
(3) $4 \times n^2$ は 4 の倍数
(4) N=$(2n-1) \times (2n+1) + 1$
(5) n は整数なので、n^2 も整数
(6) N=$4 \times n^2$
(7) n を整数とする

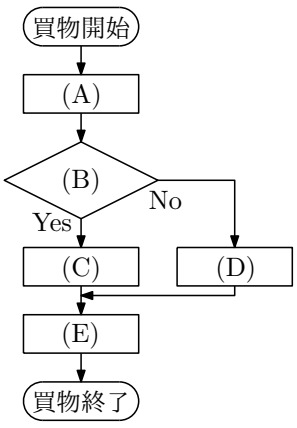


(A)＿＿＿＿ (B)＿＿＿＿ (C)＿＿＿＿ (D)＿＿＿＿ (E)＿＿＿＿ (F)＿＿＿＿ (G)＿＿＿＿

例 1.1.5. (**買い物問題**)

お弁当を買いにコンビニへ行き、ハンバーグ弁当とから揚げ弁当のうち安い方を買って家に帰る。(同額は考えない)

(1) ハンバーグ弁当を買う
(2) から揚げ弁当を買う
(3) 家に帰る
(4) コンビニに行く
(5) ハンバーグ弁当の方が安い



(A)＿＿＿＿ (B)＿＿＿＿ (C)＿＿＿＿ (D)＿＿＿＿ (E)＿＿＿＿

1.2 アルゴリズムとデータの変化

1.2.1 1〜n までの和を求めるアルゴリズム

n を自然数とし、1 から n までの和 (sum) を求めるには繰り返し (反復構造) を用いる。

$$sum = 1 + 2 + 3 + 4 + 5 + \cdots + (n-1) + n$$

を計算するには左から順に演算を行っていく。このアルゴリズムは以下のようになる。

アルゴリズム：

1. 合計用の変数 sum を 0 で初期化する
2. 合計値に加算して行く変数 i を 1 にする
3. i が n 以下の間、次の手順 4. と 5. を繰り返す
4. sum に i を加えて、その結果を sum に代入する
5. i に 1 加える。
6. sum が 1 から n までの合計値

例 1.2.1. **(n が 4 のときの例)**

n が 4 のとき、各データ (変数) の変化を表にすると以下の通りとなる。

手順	処理	sum	i	n
1	$sum = 0$	0		4
2	$i = 1$	0	1	4
3	$i \leqq n$ か? → 真 (繰り返し)	0	1	4
4	$sum = sum + i$	1	1	4
5	$i = i + 1$	1	2	4
3	$i \leqq n$ か? → 真 (繰り返し)	1	2	4
4	$sum = sum + i$	3	2	4
5	$i = i + 1$	3	3	4
3	$i \leqq n$ か? → 真 (繰り返し)	3	3	4
4	$sum = sum + i$	6	3	4
5	$i = i + 1$	6	4	4
3	$i \leqq n$ か? → 真 (繰り返し)	6	4	4
4	$sum = sum + i$	10	4	4
5	$i = i + 1$	10	5	4
3	$i \leqq n$ か? → 偽 (終了)	10	5	4
6	sum が 1 から n までの合計値	$\boxed{10}$	5	4

※ここでの $=$ は、左辺に右辺の値を代入するという意味で使用している。

♠ **補足** $sum = \dfrac{1}{2}n(n+1)$ で計算することも 1 つのアルゴリズムである。

1.2.2　配列データに順位付けのアルゴリズム

複数のデータ (N 個) を大きい順に順位をつけることを考える。例えば ($N = 5$)、

$$75 \text{ 点}, 85 \text{ 点}, 63 \text{ 点}, 59 \text{ 点}, 91 \text{ 点}$$

だとすると、点数の高い順では 3 位, 2 位, 4 位, 5 位,1 位となる。

ここで、N 個のデータに順位付けのアルゴリズムを考える。

まず、配列 (数列と思っても可)p_i と r_i ($1 \leqq i \leqq N$) を用意し、与えられた点数は p_i に格納し、点数 p_i の順位を r_i に格納する。(添え字を扱う変数 k, l も用意)

アルゴリズム：

1. 全ての r_i を 1 で初期化する
2. 添え字を保存する変数 k を 1 にする
3. k が N 以下のとき、手順 4〜8 を行う
4. 添え字を保存する変数 l を 1 にする
5. l が N 以下のとき、手順 6〜7 を行う
6. $p_k < p_l$ なら r_k を $+1$ する
7. $l = l + 1$
8. $k = k + 1$
9. 順位付け終了

例 1.2.2. **(N が 5 のときの例)** 5 人のテストの点が

$$p_1 = 75,\ p_2 = 85,\ p_3 = 63,\ p_4 = 59,\ p_5 = 91$$

のとき、順位を求めるために r_i の変化を以下に示す。

手順	処理	k	N	r_1	r_2	r_3	r_4	r_5
1	$r_i = 1$		5	1	1	1	1	1
2	$k = 1$	1	5	1	1	1	1	1
3	$k \leqq N$ か? $\to$ 真	1	5	1	1	1	1	1
4〜7	$1 \leqq l \leqq N$ で $p_k < p_l$ なら $r_k = r_k + 1$	1	5	3	1	1	1	1
8	$k = k + 1$	2	5	3	1	1	1	1
3	$k \leqq N$ か? $\to$ 真	2	5	3	1	1	1	1
4〜7	$1 \leqq l \leqq N$ で $p_k < p_l$ なら $r_k = r_k + 1$	2	5	3	2	1	1	1
8	$k = k + 1$	3	5	3	2	1	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
8	$k = k + 1$	6	5	3	2	4	5	1
3	$k \leqq N$ か? $\to$ 偽	6	5	3	2	4	5	1
9	順位付け終了	6	5	$\boxed{3}$	$\boxed{2}$	$\boxed{4}$	$\boxed{5}$	$\boxed{1}$

1.2.3　ソートのアルゴリズム

与えられたデータを、昇順または降順に並べ替え (ソート) を行うことは日常でもよく行うことである[*1]。ここでは簡単なソート方法を 2 つ紹介し、データの変化を考えてみる。

ソートのアルゴリズムは沢山あるので、各自調べてみて貰いたい。

・選択ソート

対象のデータの中から最小値 (最大値) を探し、データの先頭 (末尾) から順番に入れ替えを繰り返してソートする

・バブルソート (単純交換法)

隣り合うデータの大小を比較し、大小関係が正しくなるように入れ替えてソートする

例 1.2.3. (5 つのデータのソート)

5 つのデータ $a_1 = 75,\ a_2 = 85,\ a_3 = 63,\ a_4 = 59,\ a_5 = 91$ を昇順にソートする。

選択ソート

作業	a_1	a_2	a_3	a_4	a_5
初期	75	85	63	59	91
1	59	85	63	75	91
2	59	63	85	75	91
3	59	63	75	85	91

バブルソート

作業	a_1	a_2	a_3	a_4	a_5
初期	75	85	63	59	91
1	75	63	85	59	91
1’	75	63	59	85	91
2	63	75	59	85	91
2’	63	59	75	85	91
3	59	63	75	85	91

選択ソートの場合

まず a_1〜a_5 の中の最小値を探す。$a_4 = 59$ が最小値なので、a_1 と入れ替える。

次に、a_2〜a_5 の中の最小値を探す。$a_3 = 63$ が最小値なので、a_2 と入れ替える。

次に、a_3〜a_5 の中の最小値を探す。$a_4 = 75$ が最小値なので、a_3 と入れ替える。

最後に、a_4 と a_5 を比較する。$a_4 = 75$ が小なので、a_4 と入れ替える。(同じものの入替えになるが、アルゴリズムとしては、入れ替えを行う。)

バブルソートの場合

まず a_1 と a_2 を比較し、大小関係が正しいので入れ替えない。

a_2 と a_3 を比較すると、大小関係が正しくないので、入れ替える (1)。

さらに、a_3 と a_4 を比較すると、大小関係が正しくないので、入れ替える (1’)。

最後に a_4 と a_5 を比較すると、大小関係が正しいので入れ替えない。

これが 1 回のループとなる。このとき、入れ替えを行った事を記憶しておく。

以下、a_1 と a_2, a_2 と a_3, a_3 と a_4, a_4 と a_5 の [比較, 入替] を行い、入れ替えを行わない回まで繰り返す。

[*1] 回収した演習問題を学生番号順に並べ替える作業はなかなか大変

1.2.4　探索のアルゴリズム

探索とは、複数のデータの中から目的のデータを探すことである。目的のデータは唯一つ存在する場合と、複数または無い場合で多少アルゴリズムが異なる。ここでは、唯一つ存在する場合に限って、2 つの探索方法に対して例題を考えてみる。

・線型探索

先頭のデータからしらみつぶしに 1 つずつ確認し、探索する方法

・二分探索

探索対象のデータの中央を基準に、探索対象を狭めて探索する方法

問題 1.2.1. (数当てゲーム)

『1 から 100 までの好きな (自然) 数を 1 つ思い浮かべてください。』
質問をしながらその数を当てるためのアルゴリズムはどのようなものがあるか?

この問題では線型探索なら、1 から順に『1 ですか?』, 『2 ですか?』, ... と聞くことになる。この場合、1 回目で当たるかもしれないが、最高で 100 回探索 (質問) が必要になる。

では、二分探索ではどうなるか。例えば 21 だとする。

『50 以上ですか?』 → 『いいえ』
『25 以上ですか?』 → 『いいえ』
『13 以上ですか?』 → 『はい』
『19 以上ですか?』 → 『はい』
『22 以上ですか?』 → 『いいえ』
『20 以上ですか?』 → 『はい』
『21 ですか?』 → 『はい』　(この答えが『いいえ』なら思い浮かべた数は 20 となる。)

この探索方法では 7 回の質問で見つけ出すことが出来た。別の数を想い浮かべていたとしても二分探索では高々 7 回で見つけることが出来る。

1.2.5　その他のアルゴリズム

線型代数学で学んだガウスの消去法 (掃き出し法) は、未知数を 1 つずつ減らしていくことによって連立方程式を解くアルゴリズムである。

また、高次方程式の解を求めるニュートン法は、関数 $f(x)$ が x 軸と交わるとき、その微分 $f'(x)$ を利用して $f(x) = 0$ の解を求めるアルゴリズムである。

数学には多くのアルゴリズムがある。これらをプログラミング出来るようになることを 1 つの目標とする。

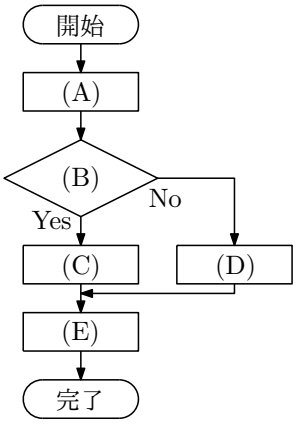
1.3 章末問題 I

課題 1.3.1. 次の文章の手順にあったフローチャートを完成せよ。

家を出発して、大学に行く。そのとき、雨が降っていればバスで行くが、降っていなければ自転車で行く。

(1) 雨が降っているか
(2) 自転車に乗る
(3) バスに乗る
(4) 家を出発
(5) 家に帰る
(6) 大学に到着

(A)＿＿＿ (B)＿＿＿ (C)＿＿＿ (D)＿＿＿ (E)＿＿＿

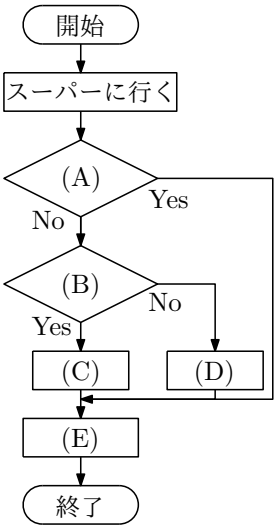


課題 1.3.2. 次の文章の手順にあったフローチャートを完成せよ。

果物を買いにスーパーに行く。りんごか、みかんのうち安い方を買うことにする。ただし、りんごとみかんは同額で無く、スーパーが休みなら家に帰ることにする。

(1) りんごを買う
(2) みかんを買う
(3) スーパーが営業している
(4) みかんよりりんごの方が安い
(5) スーパーが休み
(6) 家に帰る

(A)＿＿＿ (B)＿＿＿ (C)＿＿＿ (D)＿＿＿ (E)＿＿＿

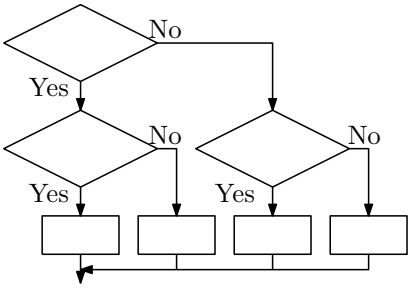


課題 1.3.3. 次の文章の手順にあったフローチャートをヒントをもとに作成せよ。

果物を買いにスーパーに行く。

りんご、みかん、バナナのうち一番安いモノを買うことにする。ただし、りんご、みかん、バナナはそれぞれ同額でないものとする。

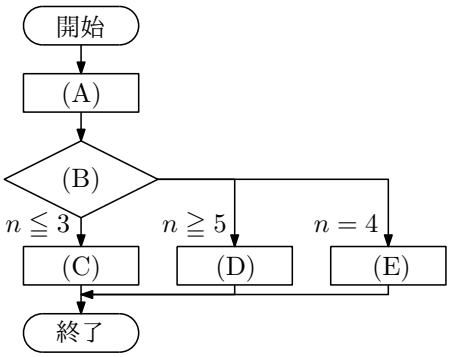
〖ヒント〗



課題 1.3.4. 次の文章の手順にあったフローチャートを完成せよ。

サイコロを振って、3 以下なら 1 つ進み、4 なら一回休み、5 以上なら 1 つ戻ることとする。

(1) サイコロのでた目が n である
(2) サイコロを振る
(3) 1 つ進む
(4) 2 つ進む
(5) 1 つ戻る
(6) スタートに戻る
(7) 一回休み

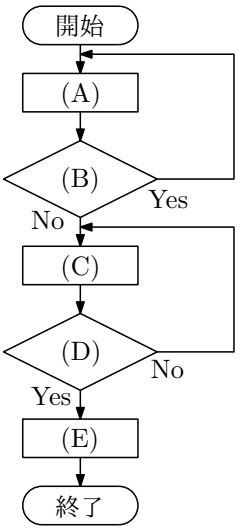


(A)　　(B)　　(C)　　(D)　　(E)

課題 1.3.5. 次の文章の手順にあったフローチャートを完成せよ。

今 9 時で微分積分学を勉強している。10 時になったら線型代数学を勉強し、12 時になったら寝る。

(1) 線型代数学を勉強する
(2) 微分積分学を勉強する
(3) 計算機とアルゴリズムを勉強する
(4) (時計を見ると) 今 $\geqq$10 時
(5) (時計を見ると) 今 $<$10 時
(6) (時計を見ると) 今 $\geqq$12 時
(7) (時計を見ると) 今 $<$12 時
(8) 寝る



(A)　　(B)　　(C)　　(D)　　(E)

課題 1.3.6. 以下の処理を用いて、例 1.1.3 のアルゴリズム (ユークリッドの互除法) に対応するフローチャートを作成せよ。

(1) b の値が最大公約数
(2) $r = 0$ ではない？
(3) $a < b$ である？
(4) a,b の値を決める
(5) b に r の値を代入する
(6) a に b の値を代入する
(7) a の値と b の値を入れ替える
(8) a を b で割った余りを r とする

課題 1.3.7. 例 1.2.1 を参考に $n = 4$ 個のデータ $a_1 = 7,\ a_2 = 5,\ a_3 = 6,\ a_4 = 9$ の合計 (sum) を求めるアルゴリズムを作成し、各データの変化を表に記せ。ただし、データが定義されていない場所は空白とすること。また、表の行は余ってもよい。

アルゴリズム：

1. _____

2. _____

3. _____

4. _____

5. _____

6. sum が a_1 から a_n までの合計値 _____

手順	処理	sum	i	a_i	n
6	sum が a_1 から a_n までの合計値				

課題 1.3.8. 100m 走の秒数の短い人から 1 位,2 位,... と順位をつける。5 人のタイムが

$$p_1 = 16.05,\ p_2 = 14.86,\ p_3 = 17.46,\ p_4 = 9.59,\ p_5 = 14.86 \quad (秒)$$

の場合に、例 1.2.2 を参考にアルゴリズムを考え、各データの変化を表に記せ。ただし、r_i は例 1.2.2 と同様に順位を表す変数である。また、表の行は余ってもよい。

処理	k	N	r_1	r_2	r_3	r_4	r_5
順位付け終了			☐	☐	☐	☐	☐

第 2 章

C 言語の基礎

2.1 初めに、基礎知識

2.1.1 ソースファイルとコンパイル

コンピューターが実行できるファイルは、コンピューターにとって理解しやすい形式になっているため、我々が見ても理解は困難である。そのため、コンピューターが実行できるファイルを作成するために、我々が見て理解できるプログラム (ソースファイル) を作成する。その後、コンパイラを用いて、コンピューターが実行できるファイルを作成する。

ソースファイルの作成 ⇒ コンパイルとリンク ⇒ 実行可能ファイル

ソースファイルを作成した後に実行可能ファイルを作成するためにコンパイルする[*1]。

コンパイルとは簡単に言うと”我々が分かる言語で書いたプログラム（ソースコード）の中身をコンピュータが処理出来る言葉に翻訳すること”である。

2.1.2 ソースファイルの名前

この講義において、ソースファイルのファイル名に用いることの出来る文字は、

ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz0123456789+-_( )

である。これらは、すべて半角であり、空白や. は使えない。また、ファイル名の後ろに拡張子 `.c` をつけることを忘れないように。

例えば、 `kadai-2-1.c` や `mondai0201.c` などは良いが、**課題 2-1.c** や `kadai 2.1.c` としたりしてはいけない。

[*1] コンパイルしたファイルや標準ライブラリをリンクする作業があるが、意識する必要が無いため省略する

2.1.3　最初の C プログラム

まず、プログラム (ソースファイル) の例をみながら少し解説を行っていく。

例 2.1.1. Hello C と表示するプログラム

```
1 #include <stdio.h>
2 int main(void)
3 {
4    printf("Hello C world¥n");
5    return 0;
6 }
```

実行結果

```
Hello C world
```

(1) 各行の左端にある数字は **行番号** を表している。プログラムに関与はしないが、見易くするために **付く** ことが多い。自分で入力してはいけない。
この講義の実習で使用する bcpad でも自動で付けてくれる。エラーが発生したときに行番号があると解りやすい。

(2) 1 行目の
`#include <stdio.h>`
は**ヘッダファイル** (`stdio.h`) をここで読み込む (インクルードする) ことを指定している。この `stdio.h` には入出力に関する関数が定義されている[*2]。
今後、必要に応じて別のヘッダファイルを読み込むことがあるが、ほぼ 100% stdio.h は読み込む。

☆ C 言語は基本単位の 1 つとして、関数と呼ばれる マトマリ で構成される。この有効範囲は { } で指定する。

(3) 2 行目で定義している main 関数はその名の通り、プログラムのメインとなるものである。main の前の int は、この関数が整数型であることを意味し、引数の void は main 関数に引き渡すデータがない (main 関数は引数を受け取らない) ことを意味している。そして、この main 関数の有効範囲は 3 行目から、6 行目である。

☆ main 関数にデータを渡す場合、以下のように引数を付けることがある。
`int main(int argc, char *argv[])`
この例 2.1.1 では、引数が `void` になっている。`void` は、C 言語において、”何もない”ことを意味するデータ型である。

[*2] 名前の由来は標準入出力を意味する「Standard I/O(Input/Output)」の略だと思う。

(4) 関数は沢山定義することが出来るが、必ず main 関数を 1 つ定義しなければならない。また、既に (stdio.h 内で) 定義された関数もたくさんある。
例.) fclose, fopen, printf, scanf, getc, getchar など

(5) この main 関数の中には 1 つの**文**があり、それが 4 行目である。1 つの文の終わりにはセミコロン (;) をつける。
この printf() も関数で、() の中を表示させる関数である。

(6) 5 行目は、呼び出し側に戻り値 0 を返している。
main 関数を呼び出すのは (通常)MS Windows や Linux などの OS である。OS は値 0 を受け取ると正常終了と判断し、0 以外の値は異常終了と判断することが多い。

2.1.4 プログラムのルール

トークン プログラムはトークンと呼ばれる最小単位のテキストの集まりで書かれている。トークンは英語における英単語に相当するもので、それ以上は分解できない (意味が変わる)。

トークンは、次の 6 つに分類される。

・キーワード	予約済みの名前
・識別子	データの位置などを識別するための名前、例えば関数名
・定数	固定的な値、例えば 1 や 0.5 などの数値
・文字列リテラル	二重引用符”で囲まれた文字列
・演算子	計算に用いる記号、例えば+や-など
・区切り子	[ ] ( ) { } ; , #など

演算子や区切りは続けて書いても良いが、キーワード、識別子や定数などは空白類文字で区切らなければならない。

セミコロン プログラムにおいて、1 つの文の終わりにはセミコロン ; を付ける。例 2.1.1 では main 関数の中に 2 つの文が書かれていることが解る。

カッコ カッコは開いた数と、閉じた数が同じでなければならない。また、対応するカッコの順番も注意しなければならない。

例 2.1.2. (カッコに関するエラー)

```
#include <stdio.h>
int main(void){
  printf("Sample File¥n"});
  return 0;
}
```

2.1.5　プログラムを読みやすくする

プログラムは; までが 1 文となるため、1 行に 2 文, 3 文を書いても問題無い。例えば、例 2.1.1 を

```
#include <stdio.h>
int main(void){printf("Hello C world¥n");return 0;}
```

と書いても正常に動作する。

しかし、1 行にまとめて書くと関数の有効範囲や文の切れ目が解り辛くなる。

♡ **注意!**　ただし、# から始まる記述だけは単独行に書く決まりである。

2.1.6　字下げ、インデント

例 2.1.1 で、4 行目と 5 行目が**字下げ (インデント)** されている。これは関数の有効範囲を見易くするためである。字下げを行わないと、関数の有効範囲等の { } で囲まれている部分が解りづらい。さらに、{ } は二重カッコ、三重カッコと使う場合や、省略することがあるので、最初から字下げをする習慣を付けておくのが良い。

例 2.1.3. 字下げを行わない例

```
#include <stdio.h>
int main(void)
{
printf("Hello C world¥n");
return 0;
}
```

2.1.7　空白類文字

プログラムを見易くするために字下げと空行を入れている。C 言語のコンパイラは空白文字、改行文字、タブ文字は全て空白文字として扱う。(以下、**空白類文字**という。)

例 2.1.4. 一般的なプログラムの例

```
#include <stdio.h>
int main(void){
    int i,sum;
    sum=0;
    for(i=1;i<=10;i++)
        sum=sum+i;
    printf("1 から 10 までの和は %d です。¥n",sum);
    return 0;
}
```

空白類文字はいくつ連続していも 1 つの空白と見なす。したがって、見易いように空白類文字を使ってよい。例えば、例 2.1.4 の 3 行目と 4 行目を

```
int      main      (       void       )       { int i,
  sum  =  0  ;
```

のように (空白を足したり改行を消したものに) 書き換えてもよく、また、トークンの切れ目に改行を入れて書いてもよい。しかし、

```
intmain(void) や int main(vo  id)
```

のように、必要な空白を消して 2 つのトークンを結合したり、トークンの途中で空白類文字を入れてはいけない。

♡ **注意!** 空白類文字に 全角スペース は含まれない！

♠ **補足** 空白をノートやプリントで明記する場合、␣を用いることがある。例えば、

```
int      main      (       void       )           { int i,sum  ;
```

の空白の数を明記する場合

```
int␣␣␣␣␣␣main␣␣␣␣␣␣(␣␣␣␣␣␣␣void␣␣␣␣␣␣␣)␣␣␣␣␣␣␣␣␣␣{␣int␣i,sum␣␣;
```

のように書く。見慣れない記号かもしれないが、␣は半角スペース 1 つを表す記号であり、半角スペース 1 つを入力すれば良い。

2.1.8 コメント

プログラムを見易くするために、プログラムのコメント (説明) をプログラム中に書くことがある。コメントはコンパイルでは無視される。

1 行のコメントの場合 // を使い、複数行にまたがるときは /* ～ */ を使う。

例 2.1.5. (コメント)

```
#include <stdio.h>
int main(void) // main 関数を定義します。
{
    printf("Hello C¥n");
    return 0;
/*
ここと
ここはコメントです。
*/
}
```

コメントは、後から自分でも見直すときや、エラーの発見に役立つので長めのプログラムを作成するときは書くように習慣づけるとよい。

2.1.9　行番号

これは、自分で付けるものではなく、エディタが自動で付加してくれる。書籍やプリントなどで行番号を記載すると後の説明が解り易くなる場合がある。

プログラムを入力するときに、行番号を自分で入力してはいけない。

例 2.1.6. (行番号も記載して説明する例)

```
 1 #include <stdio.h>
 2 int main(void) // main 関数を定義します。
 3 {
 4     printf("Hello C¥n");
 5     return 0;
 6 /*
 7 ここと
 8 ここはコメントです。
 9 */
10 }
```

このプログラムでは、2 行目の後半にコメントがあり、4 行目と 5 行目は字下げされている。また、6 行目から 9 行目はコメントとなっているため、プログラムに影響はない。
のように説明が容易である。

2.1.10　エラー

実際にプログラミングをしていると、何度もエラーに出会う。エラーが起きた場合は、エラーの場所を探して直すしかない。その場合、メッセージが表示されているので参考にする。また、エラーの原因として多いものを覚えておく。

例 2.1.7. (セミコロン (;) が無い例)

メッセージ

```
エラー E2379 hello.c 5: ステートメントにセミコロン ( ; ) がない (関数
main )
警告 W8070 hello.c 7: 関数は値を返すべき (関数 main )
*** 1 errors in Compile ***
```

実際のエラーでは 1 つのミスでも 2 つ以上のエラーとして表示されることがある。

♠ **補足**　警告はエラーのようにコンパイルが出来ないほどではないが、プログラムにおいて適切とは言えない記述があることを示している。

エラーの原因で多いもの

(i) { と }、(と)、” と ” の整合性が合っていない。
[対処] どの開きカッコと、閉じカッコが対応しているか 1 つずつ確認する。

(ii) セミコロン ; が無い。
[対処] 各行末を確認する。セミコロンでなく、コロン : になっている場合がある。

(iii) ファイル名が正しくない。
[対処] ソースファイルのファイル名を確認する。特に、全角文字を使用していたり、拡張子との区切り以外でドット (.) を使用していないか確認する。

(iv) 綴りのミスや、全角半角のミス。
(a) 始めの内多い綴りミスは `stdio.h`.
[対処] 1 文字ずつ確認する。
(b) ”, { などの記号やスペースが全角になっているミス。
[対処] 全角の”や｛で検索をかけてみる。

全角のスペースは半角のスペース 2 個と見分けがつかない。エラーがおきたとき、本当に見つけにくい間違いである。

しかし、応用数学科計算機室にインストールされている bcpad の場合は、全角スペースは□で表示されるのでこのミスは少ない。他のエディタを使用する場合は、注意が必要である。

2.1.11 関数

関数の宣言 (関数の定義) について詳しくは説明しないが、ここでは引数と戻り値を理解するために 1 つだけ例をあげておく (詳しくは計算機とアルゴリズム II で)。

例 2.1.8. 関数と戻り値の例

```
int sum(int a, int b)
{
   return a+b;
}
```

この例では、関数 `sum()` に 2 つの値を入れ、その和を戻り値として返す例である。実際の使用例は以下のようなものである。

```
a=sum(10,20);
z=sum(x*x,y*y);
p=sum(sum(2,3),5);
```

詳しくはここで述べないが、直感的に解るだろう。

2.1.12　printf 文 I

printf 文は標準出力（通常は画面) に文字列や数値 (データ) を出力する関数である。

書式

printf(書式指定文字列, データの並び);

書式指定文字列とは、二重引用符で囲まれた一般文字列または変換指定をいう。

一般文字列とは、一般的に使用される文字列 (abc, あいうえお, 日本語 など) や 改行を意味する ¥n などである。

変換指定は、%+ 変換指定子 で構成されている。詳しくは 2.2.7 で述べる。

例 **2.1.9.** printf 文の例 1.

```
1 #include <stdio.h>
2 int main(void){
3    printf("Hello C world");
4    return 0;
5 }
```

実行結果

```
Hello C world
```

printf() 文の引数にある " " で囲まれた書式指定文字列 (の中の一般文字列) が表示されている。この一般文字列には全角文字 (日本語や記号) も含めることができる。

例 **2.1.10.** printf 文の例 2.

```
1 #include <stdio.h>
2 int main(void){
3    printf("こんにちは C の世界へ");
4    return 0;
5 }
```

実行結果

```
こんにちは C の世界へ
```

♡ **注意!**　特殊な記号はそのままでは表示できない。例えば

%, **¥**, **"**

の記号を表示させる場合、以下のように入力しなければならない。

% は **%%** もしくは **¥%**,　**"** は **¥"** ,　**¥** は **¥¥**

例 **2.1.11.** 特殊記号の表示

```
1 #include <stdio.h>
2 int main(void){
3    printf(" 10%% ¥"¥¥500¥" ");
4    return 0;
5 }
```

実行結果

```
10% "¥500"
```

printf() 文を 2 つ書き、2 行の一般文字列を表示させたいと思った場合、実行結果において、この 2 つの一般文字列は繋がって表示される。すなわち、改行はされない。

例 **2.1.12.** 文字を表示させる例 1.

```
1 #include <stdio.h>
2 int main(void){
3    printf("こんにちは");
4    printf("C の世界！");
5    return 0;
6 }
```

実行結果

```
こんにちは C の世界！
```

C 言語の出力においては、”改行を表示させない限り”、続けて表示される。”改行を表示させる”には、¥n を表示させればよい。

例 **2.1.13.** 文字を表示させる例 2.(改行付)

```
1 #include <stdio.h>
2 int main(void){
3    printf("こんにちは ¥n");
4    printf("C の世界！");
5    return 0;
6 }
```

実行結果

```
こんにちは
C の世界！
```

♠ **補足** 例 2.1.13 の 3 行目と 4 行目をまとめて

```
3    printf("こんにちは ¥nC の世界！");
```

としても、例 2.1.13 と同様の実行結果を得られる。

2.2　変数と定数

2.2.1　名前 (識別子)

プログラムの中で使用する名前 (変数名、関数名、記号定数名など) には、使用できる文字に決まりがあり、以下に挙げる文字で、半角文字に限られる。(ファイル名で使える文字とは異なることに注意)

名前 (識別子) に使用可能な文字

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9 _

さらに、名前の付け方には以下の決まりがある。

- **英文字 または _ で始めなければならない。**
- **長さに制限はない (が先頭から最低でも 31 文字が識別対象になる)。**
- **予約語を識別子に使うことは出来ない。例えば、変数名に for は使えない。**
- **大文字と小文字は区別される。data, DATA, Data は異なる名前である。**

名前の長さはコンパイラによって解釈が異なるが、意味が解らない省略形より、長くても解りやすくすることが望まれる。例えば、`md` などとするより、`MaxData` とする。`md` では、`MinData` や `MonthData` など混乱が生じる。したがって、31 文字以内なら解りやすい名前にすることが望ましい。

また、プログラム中で include したファイルに定義されている関数名も名前として使用することは出来ない。例えば、`stdio.h` を読み込んだ場合、`printf` や `scanf` などは使用できない。

例 2.2.1. 名前の例

`kadai`	正しい
`0510kadai`	誤り (先頭に数字を使用できない)
`kadai0510`	正しい
`kadai-0510`	誤り (-は使えない)
`FOR`	正しい (`for` は予約語なので使えない)
`file_name`	正しい (スペースの代わりに_を使うことは多い)
`FileName`	正しい (読みやすくするために大文字を使うことはよくある)

2.2.2 予約語

C 言語では次のモノが**予約語**に定められている。予約語は名前として使用出来ない。

```
 auto    double     int      struct     break     else     long    switch
 case     enum    register  typedef     char    extern  return   union
const    float     short    unsigned  continue    for    signed    void
default  goto     sizeof    volatile     do       if     static   while
```

2.2.3 定数

C 言語で扱われるデータは”定数”と”変数”に分けらる。もちろん、”定数”は値を変えないもので、”変数”は任意の値をとりうるものである。

```
dt=100;
```

と書けば、dt は変数であり、100 は定数である。

♠ **補足説明** C 言語において、等号記号 = は”等しい”という意味ではなく、”代入”を意味する。上の場合、変数 dt に定数 100 を代入している。

2.2.4 整数定数と浮動小数点定数

整数定数は小数部のない数値定数である。また、整数定数には 10 進数、8 進数、16 進数の表記があり、定数の大きさと符号の有無を示す接尾語 (u, U, l, L, LL) がある。ただし、 この講義の実習では LL(long long) は使えない。また、L 有り無しで変化はない。

例 2.2.2. a に 30 を代入する例。

```
a=30;
a=036;
a=0x1e;
a=30u;
```

浮動小数点定数は小数部を持った数値定数である。また、e や E を用いて指数形式で表すこともできる。

例 2.2.3. a に 123.456、b に 0.000123 を代入する例。

```
a=123.456;
a=1.23456e+02;
b=0.000123;
b=1.23e-04;
```

♡ **注意!** 浮動小数点定数を代入するとき、c=100f は誤り。c=100.0f とすべきである。また、0.00000000000123 などは指数形式を用いないと定義できない。

2.2.5　文字定数と文字列リテラル

文字定数

1 つの文字をシングルクォーテーションで囲み (例えば 'A')、その文字の文字コードの値を**文字定数**という。'A' は A の文字コードの数値で 65 である。(よって、文字の演算も出来る。参照例 2.2.12)

例 2.2.4. 以下の (1) と (2) は同じ意味である。

```
int c;
c='A';  // (1)
c=65;   // (2)
```

最初の `int c;` は、"整数 (integer) の変数 `c` を用意する" とだけ、(今は) 理解しておいてほしい (2.3.1 変数の宣言で説明)。

文字列リテラル

文字列を、ダブルクォーテーションで囲んだものを**文字列リテラル**といい、例えば"moji"や"文字列"である。ただし、日本語 (2 バイト文字) の扱いはコンパイラによって異なるので注意が必要である。

♡ **注意!**　C 言語では 文字列を代入するとき、`a="moji"`のように扱えない。文字列は配列を使い、以下のように扱う。

例 2.2.5. 文字を代入する方法

```
char txt[10]="Hello";  // 文字配列 txt に文字列"Hello"を初期設定する
strcpy(txt, "Hello");  // 文字配列 txt に文字列"Hello"をコピーする
```

文字列の扱いについては 3.1.2 および 5.2 で説明する。

2.2.6　その他の定数機能

エスケープシーケンス

`printf()` 関数で用いた `¥n` は改行を意味する文字である。この文字は先頭に `¥`があるため、本来の文字 `n` を離れ (エスケープし)、特別な意味の文字となる。このような文字を**エスケープシーケンス**と呼ぶ。よく使うエスケープシーケンスは、

`¥n` (改行), `¥t` (水平タブ), `¥xhh` (16 進数)

である[*3]。`a=0x32` と `a='¥x32'` は同じ意味であるが、後者は 2 桁でしか使えない。

[*3] これ以外は教科書の 31 ページを参照

例 **2.2.6.** 水平タブを表すエスケープシーケンスを用いた例

```
printf("abc¥tklm¥nnop¥ns¥tt¥tu¥n");
```

実行結果

```
abc     klm
nop
s       t       u
```

例 **2.2.7.** 16 進数を表すエスケープシーケンスを用いた例

```
printf("%d¥n",'¥x30');
```

実行結果

```
48
```

これは、16 進数で 30 と表される数を 10 進数 (%d) で表示させている。

記号定数と関数（形式) マクロ

任意のデータを、意味のある名前で扱うために**記号定数**という機能がある。例えば

```
#define MaxData 3286
```

としておく。(この場合は最後に ; を付けない) これは、プログラム中に MaxData が出てきたら 3286 に置きかえるということである。

プログラムの中で同じ数値が何か所もあり、度々変更する場合など記号定数を用いれば 1 箇所を替えるだけで反映されるメリットがある。

記号定数の定義と同様に、#define を用いて関数（形式) マクロを定義することができる。関数 (形式) マクロは、定義の形が関数を呼び出しているように見えるため、このように呼ばれている。

```
#define 関数マクロ名（引数） 処理内容
```

であるが詳しい説明は省略するが、1 つ例のみ紹介する。

```
#define average(a,b) (a+b)/2
// 行末に ; はない。
```

const 付き変数

変数を const 付きで宣言することが出来る。この変数は宣言のときに値を定め、これ以降は値を変更することは出来ない。例えば、

```
const int dt=100;
```

と書けば、dt も 100 も定数である。

// あまり使わないと思う。

2.2.7　printf 文 II

2.1.12 で紹介した printf 文の書式は

printf(書式指定文字列, データの並び);

であった。書式指定文字列とは、二重引用符で囲まれた一般文字列または変換指定であり、

一般文字列：abc , 123, 日本語, あいうえお, ¥t, ¥n など
変換指定：%d や %s など [%+ 変換指定子] で構成

である。変換指定には以下のようなものがある。これらは、データの並びにある値を変換して出力するためのものである。

変換指定	意味
%o	8 進数で出力
%d	10 進数で出力
%x	16 進数で出力 (英小文字)
%X	16 進数で出力 (英大文字)
%ld	long型整数値を 10 進数で出力
%f	浮動小数点として出力
%e	指数形式で出力 (英小文字)
%E	指数形式で出力 (英大文字)
%c	文字として出力
%s	文字列として出力
%p	アドレスを出力

♡ **注意!**　上記の o や d が変換指定子である。

書式指定文字列の中に、変換指定を複数指定した場合、データの並びの左から順に処理をしていく。

例 2.2.8. 数値を出力するプログラムの例.

```
#include<stdio.h>
int main(void){
  printf("10 進数で表す %d は、8 進数で表すと %o で、
            16 進数で表すと %x です。¥n",30,30,30);
  return 0;
}
```

実行結果

```
10 進数で表す 30 は、8 進数で表すと 36 で、16 進数で表すと 1e です。
```

変換指定子のオプション

数の表示において、桁数や精度を指定するため以下のオプションが使用できる。

% フィールド幅指定子 . 精度指定子 変換指定子

フィールド幅指定子 で数値の出力幅を指定する。

精度指定子 で浮動小数点の小数以下の桁数を指定する。

桁数をそろえて表示する場合には有用である。

例 **2.2.9.** オプションを用いた表示例 1

```
#include<stdio.h>
int main(void){
    int a=20;               // 2.3.1 変数の宣言で説明
    double da=12.3456789;   // 2.3.1 変数の宣言で説明
    printf("a=%d,¥n",a);    // %d: 標準用法 10 進数で出力する
    printf("a=%8d,¥n",a);   // %8d:8 桁の 10 進数で出力する 先行 0 はない
    printf("a=%08d,¥n",a);  // %08d:8 桁の 10 進数で出力する 先行 0 がある
    printf("a=%-8d,¥n",a);  // %-8d: 左寄せで 8 桁の 10 進数を出力
    printf("da=%f,¥n",da);  // %f: 標準用法 浮動小数を標準桁数で出力
    printf("da=%10.4f,¥n",da);
        // %10.4f: 浮動小数を (小数点も含めて)10 桁、小数点以下 4 桁出力する
    printf("da=%10.4f¥n",da*(-1));  //符号も 10 桁に含まれる
    return 0;
}
```

実行結果

```
a=20,
a=      20,
a=00000020,
a=20      ,
da=12.345679,
da=   12.3457,
da=  -12.3457
```

例 **2.2.10.** オプションを用いた表示例 2 (桁数を揃えたい)

```
#include<stdio.h>
int main(void){
    int y=2023,m=5,d=9;
    printf("ファイル名は %d%02d%02d.c とする。¥n",y,m,d);
    return 0;
}
```

実行結果

```
ファイル名は 20230509.c とする。
```

2.2.8　演算子 I

算術演算子

C 言語でいろいろな演算子が扱えるが、ここでは一般的によく使う算術演算子 (通常の和、差、積、商、余り) の紹介をする。

演算子	種類	例	意味
+	和	a+b	a と b を加える
-	差	a-b	a から b を引く
*	積	a*b	a と b をかける
/	商	a/b	a を b で割った商
%	余り	a%b	$(a, b$ が int 型の場合$)a$ を b で割った余り

% は a,b が浮動小数型ではエラーとなる。

例 2.2.11. 算術演算の例

```
int a=10,b=3;
printf("a+b=%d¥n",a+b);
printf("a-b=%d¥n",a-b);
printf("a*b=%d¥n",a*b);
printf("a/b=%d¥n",a/b);
printf("a%%b=%d¥n",a%b);
```

実行結果

```
a+b=13
a-b=7
a*b=30
a/b=3
a%b=1
```

例 2.2.12. 文字の演算も出来る。

```
#include <stdio.h>
int main(void){
   char c;  // 2.3.1 変数の宣言で説明
   int n;
   c='A';
   n=5;
   printf("%c の %d 文字後は %c です。¥n",c,n,c+n);
   return 0;
}
```

実行結果

```
A の 5 文字後は F です。
```

2.3 データ型と演算子

2.3.1 変数の宣言

C 言語で何かデータを保存しておく場合、**変数**を使用する。変数はデータを入れる**箱**のようなものであり、変数を使うためには**変数宣言**をしなければならない。

例 2.3.1. 変数宣言

```
int x,y;
int n=10;
double a,b;
char c;
```

それぞれの変数宣言では、変数 x，y，n を整数型、変数 a，b を浮動小数点型、変数 c を文字型で変数宣言している。これらの型を**データ型**と呼び、次のような型がある。

2.3.2 データ型

C 言語で用いられる主なデータ型は次のようなものがある。

データ型	バイト幅例	表記範囲例
char	1	−128〜127
unsigned char	1	0〜255
(signed) int	4	−2147483648〜2147483647
unsigned (int)	4	0〜4294967295
short (int)	2	−32768〜32767
unsigned short (int)	2	0〜65535
long (int)	4	−2147483648〜2147483647
unsigned long (int)	4	0〜4294967295
float	4	約 10^{-37}〜10^{38}, 有効桁数 6 桁
double	8	約 10^{-307}〜10^{308}，有効桁数 15 桁

♡ **注意!** ただし、バイト幅 (表記範囲) は各コンパイラによって異なる。

自分の環境でのバイト幅を知るためには `printf("%d¥n", sizeof(char));` や `printf("%d¥n", sizeof(int));` とすればよい。

C 言語において char 型は文字型とも言われているが、実は1 バイトの整数型である。

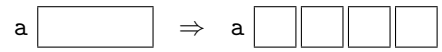
♠ **補足** 1 バイトの整数型なので扱える数は 0〜255 である。英数文字を 1 文字表すには十分であるが全角文字 (日本語) を表すには足りない。そこで、2 バイトで 1 文字を表すが、この講義では変数のデータに日本語は使えないものとする。

講義以外で使用しようと思うときは、使うコンパイラが”表”という文字を正しく扱えるか確認して使用するとよい。

2.3.3　変数の扱い

変数はデータを入れる”箱”のようなものと言ったが、その箱のサイズを変数宣言で決める。例えば、上記の `int a;` では、メモリ内に変数 `a` のための 4 バイトのメモリが確保される (`a` という名前の箱が作られ、そのサイズが 4 バイトとなる)。

`int a;` で箱を確保すると、その箱は 1 バイトの箱を 4 つ集めたものとなる。



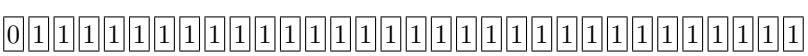
ただし、1 バイトは 8 ビット (すなわち、0 または 1 が 8 個) なので、32 個の 0 または 1 が入る箱となる。すなわち、変数宣言された変数が int 型なら

のように構成される。このとき、左端の は**符号ビット**と呼ばれ、0 なら正、1 なら負となる。データとして扱えるのはその右側からの 31 ビットとなる。これが、`unsigned int` なら符号ビットの場所もデータとし扱われる。

例えば、int 型で

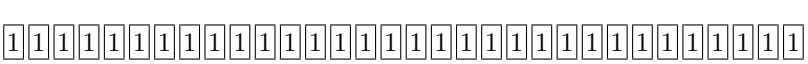
は 0 である。この変数に +1 をすると、

となり、値は 1 である。これを繰り返し



となった値が 2147483647 である。これに +1 をした

は −2147483648 である。そして、



が −1 を表している。

負の数はビット反転し、+1 したものである。例えば、1 は 0 0 0 0 ..... 0 1 であり、ビット反転をすると 1 1 1 1 ..... 1 0 となる。これに +1 すると 1 1 1 1 ..... 1 1 であり、これが −1 を表している。1 0 0 0 ..... 0 1 が −1 と思えるかもしれないが、間違えないように。

2.3.4 演算子 II

ビット演算子

ビット演算子は 2 進数で表したとき各ビット同士の演算である。

演算子	種類	例	意味
&	ビット毎の AND	a&b	a と b のビット単位の論理積
|	ビット毎の OR	a|b	a と b のビット単位の論理和
^	ビット毎の排他的 OR	a^b	a と b のビット単位の排他的論理和
>>	右シフト	a>>2	a を 2 ビット分、右へシフト
<<	左シフト	a<<3	a を 3 ビット分、左へシフト
~	ビット単位の反転	~a	a の各ビットを反転

論理積、論理和、排他的論理和は下表の通りで、a=0,b=1 なら a&b=0,a|b=1 となり、a=3,b=5 なら a&b=1,a|b=7,a^b=6 となる。ビットごとの演算であることに注意。

a&b		b	
		0	1
a	0	0	0
	1	0	1

a|b		b	
		0	1
a	0	0	1
	1	1	1

a^b		b	
		0	1
a	0	0	1
	1	1	0

例えば、a=3, b=5 なら

$$\texttt{a} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}, \quad \texttt{a\&b} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}$$
$$\texttt{b} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}, \quad \texttt{a|b} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{1}$$
$$\texttt{a\^{}b} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{0}$$

である。

また、シフト演算子は a>>1; や a<<2; のように使い、前者は a を 2 で割る、後者は a を 4 倍することである。

例えば、a=11 すなわち $\texttt{a} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{1}$ のとき、

$$\texttt{a>>1} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{0}\boxed{1}, \quad \texttt{a<<2} = \boxed{0}\boxed{0}\cdots\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}$$

である。負の数の右シフトがどうなるか、各自で確認してもらいたい。

単純代入演算子

単純代入演算子は代入を意味し、=を使う。この記号=は”等しい”を表さない。

演算子	種類	例	意味
=	代入	a=b	a に b の値を代入する

従って、a を変数とした場合、a=3 は正しいが、3=a は誤りである。

例 **2.3.2.** 代入の例

```
1 int a;
2 double b;
3 a=2.7182;   //a の値は 2 となる。
4 b=11.5;     //b の値は 11.500000 となる。
5 a=b;        //a の値は 11 となる。
6 b=a;        //b の値は 11.000000 となる。
```

この例では 3, 5 行目は警告が出る。警告なので支障はないが、このような場合は**キャスト演算子**を用いる。

キャスト演算子

この演算子はデータ型を変換するときに使用する。例えば、`a=(int)b;` とすることで、`b` の値を `int` 型 (整数型) に変換してから `a` に代入を行う。

逆に、`c=(double)d;` とすることによって、`d` の値を `double` 型に変換してから `c` に代入することも出来る。

例 **2.3.3.** キャスト演算子の例 1

```
#include<stdio.h>
int main(void){
    int a=3,b=5,c=2;
    printf("3 つの平均は %d です。¥n",(a+b+c)/3);
    printf("3 つの平均は %f です。¥n",(double)(a+b+c)/3);
    return 0;
}
```

実行結果

```
3 つの平均は 3 です。
3 つの平均は 3.333333 です。
```

例 **2.3.4.** キャスト演算子の例 2

```
#include<stdio.h>
int main(void){
   int a;
   double c=3.14159,d;
   a=(int)100*c;
   d=(double)a/100;
   printf("%f の小数点第三位以下を切り捨てると %f です。¥n",c,d);
   return 0;
}
```

実行結果

```
3.141590 の小数点第三位以下を切り捨てると 3.140000 です。
```

インクリメント・デクリメント演算子

この演算子は非常によく使う演算子でありながら、やっかいな演算子でもある。

演算子	種類	例	意味	結合性
++	インクリメント演算子	a++	a に 1 加える	左
--	デクリメント演算子	a--	a から 1 引く	左
++	インクリメント演算子	++a	a に 1 加える	右
--	デクリメント演算子	--a	a から 1 引く	右

この演算子は前置か後置かで、動作が異なる。例えば、

1) a++, ++a　　a=a+1 と同じ意味
2) c=a++;　　まず、c=a; を行い次に a++; をする
3) c=++a;　　まず、++a; を行い次に c=a; をする

となる。

例 2.3.5. インクリメント・デクリメント演算子の例

```
 1 #include<stdio.h>
 2 int main(void){
 3     int a,b,c,d,e;
 4     a=10;
 5     b=++a;      //a に 1 加えた後、b に代入する。(a=11, b=11)
 6     c=a++;      //c に a の値を代入した後、a に 1 加える。(a=12, c=11)
 7     d=--a;      //a から 1 引いた後、d に代入する。(a=11, d=11)
 8     e=a--;      //e に a の値を代入した後、a から 1 引く。(a=10, e=11)
 9     printf("a=%d, b=%d, c=%d, d=%d, e=%d¥n",a,b,c,d,e);
10     return 0;
11 }
```

実行結果

```
a=10, b=11, c=11, d=11, e=11
```

2.3.5 演算子の結合優先順位

結合優先順位を間違えると、思った値と異なる結果となる。また、コンパイラによって優先順位が異なる場合もあるので、省略ばかりしない方が良いこともある。

高	後置インクリメント・デクリメント演算子	++, --
	前置インクリメント・デクリメント演算子	++, --
	乗除演算子	*, /, %
	加減演算子	+, -
	シフト演算子	>>, <<
低	代入演算子	=

例 **2.3.6.** 各種演算の例 1

```
#include<stdio.h>
int main(void){
    int a,b;
    a=10;
    printf("(1)=%d,(2)=%d¥n",a++,++a);
    printf("(3)=%d¥n",a);
    printf("(4)=%d,(5)=%d¥n",++a,a++);
    printf("(6)=%d¥n",a);
    a=10;b=10;
    printf("(7)=%d,(8)=%d¥n",a++,++b);
    printf("(9)=%d,(10)=%d¥n",a,b);
    printf("(11)=%d,(12)=%d¥n",++a,b++);
    printf("(13)=%d,(14)=%d¥n",a,b);
 return 0;
}
```

実行結果

```
(1)=11,(2)=11
(3)=12
(4)=14,(5)=12
(6)=14
(7)=10,(8)=11
(9)=11,(10)=11
(11)=12,(12)=11
(13)=12,(14)=12
```

例 **2.3.7.** 各種演算の例 2

```
#include<stdio.h>
void main(void){
    int s;
    double a,b;
    s=1234567890; a=1234567.890; b=0.123456789;
    printf("(1)=%f,(2)=%f¥n",a+b,a-b);
    printf("(3)=%e,(4)=%e¥n",a*b,a/b);
    printf("(5)=%10.4f,(6)=%8.3f¥n",a*b,a/b);
    printf("(7)=%d,(8)=%d¥n",s,s*s*s);
    printf("(9)=%f,(10)=%f¥n",4/3*3.14,3.14*4/3);
}
```

実行結果

```
(1)=1234568.013457,(2)=1234567.766543
(3)=1.524158e+05,(4)=1.000000e+07
(5)=152415.7875,(6)=10000000.000
(7)=1234567890,(8)=-1577654328
(9)=3.140000,(10)=4.186667
```

2.4 章末問題 II

課題 2.4.1. 自分の学生番号と氏名、OUS-id を画面表示するプログラムを作成せよ。

例えば S24M999 理大 太郎 (s24m999ab@ous.jp) の場合以下のようになる。

実行結果

```
学生番号:S24M999
氏名: 理大 太郎
OUS-id:s24m999ab@ous.jp
```

♠ **補足** 幾つかの全角文字は正しく表示されず、エラーとなる場合がある。

例えば、“**表現**” はエラーがでる。これは、“表現” を示すシフト JIS のコードが [95 5c 8c bb] であり、この [5c](10 進数で表せば 92) が **¥**の文字コードのため、コンパイラがこの [5c] を **¥n** や **¥t** のようなエスケープシーケンスだと判断し、処理してしまうためである。

このことを回避するために、“**表¥現**” のように余分な **¥**を記述する方法がある。

以下同様：ソ噂浬欺圭構蚕十申曾箪貼能表暴予禄兎喀媾彌拿杤歔滴奮秉綵臂藹觸軆鐔饅鷭

課題 2.4.2. 以下の実行結果となるように、以下の空欄を一般文字列でうめてプログラムを完成させよ。(空欄に printf など文は入らない。)

```
#include<stdio.h>
int main(void){
    printf("                                    ");
    return 0;
  }
```

実行結果

```
|~^/"'‘_¥*.,:;
@&!#$<>[](){}?
```

♠ **補足** なお **¥**は環境によっては \(バックスラッシュ) と表示される。

課題 2.4.3. 次の文面を画面表示するプログラムを作成せよ。ただし、”や % など記号と数字は半角で表示させ、漢字, 仮名, 句読点のみ全角で表示させること。

実行結果

```
1 個 ¥350 のソフトクリームを 5 個買うと
   "20% 引き"
になる。このソフトクリームを 5 個買うと、いくらでしょう?
ただし、消費税は 10% とする。
```

課題 2.4.4. 次の実行結果を画面表示するプログラムを作成せよ。

実行結果

```
1 #include <stdio.h>
2 int main(void){
3    printf("Hello C world¥n");
4    return 0;
5 }
```

課題 2.4.5. 次の実行結果を表示させるプログラムを作成せよ。ただし、S は*が 0 個、A は 2 個、B は 40 個、C は 24 個、D は 9 個、E は 6 個である。

実行結果

```
          テストの結果（人数）
   |    5   10   15   20   25   30   35   40
---+----+----+----+----+----+----+----+----+---
 S |
 A |**
 B |****************************************
 C |************************
 D |*********
 E |******
```

プリントでは 上下の|が繋がっていないが、画面上では繋がっているかもしれない。
ここは気にしないで欲しい。

課題 2.4.6. 次の実行結果を表示させるプログラムを作成せよ。

実行結果

```
   ／) ／)
  /  ⌒   ＼
  | ●_ ●  |/＼
 (○ ～ ○|  ／
 /        |<
 |     L/  |/
```

課題 2.4.7. 2 の a 乗を表示させるプログラムを作成せよ。ただし、$2 \leqq a \leqq 30$ とし、4 行目で a の値を変更しても正しく表示されるプログラムにすること。

```
#include<stdio.h>
int main(void){
   int a;
   a=30;
   printf("2 の %d 乗は %d です。",                    );
    return 0;
  }
```

実行結果

```
2 の 30 乗は 1073741824 です。
```

課題 2.4.8. 7〜9 桁の自然数 a を定義し、その数を 3 桁ずつ区切り、間に , を入れて表示するプログラムを作成せよ。

ただし、a の値を変更しても正しく表示されるプログラムにすること。

実行結果

```
a を 12321012 とします。
カンマ区切りした a は 12,321,012 です。
```

課題 2.4.9. 以下の実行結果となるように次のプログラムの空欄に当てはまるものを答え、プログラムを作成せよ。ただし、(i) と (ii) は同じ文が入る。

```
#include<stdio.h>
 int main(void){
   int a;
   char c;
   c='G'; a=5;
   printf(          (i)          );
   c='P'; a=3;
   printf(          (ii)         );
   return 0;
 }
```

実行結果

```
G の 5 個後は L で、G の小文字は g です。
P の 3 個後は S で、P の小文字は p です。
```

課題 2.4.10. 2 つの整数 a,b に対して実行結果にある演算を表示するプログラムを、以下の空白の場所に文を追加し完成させよ。

ただし、a,b の値を変更しても正しく表示されるプログラムにすること。

```
#include<stdio.h>
int main(void){
    int a,b;
    a=10;b=4;

    return 0;
}
```

実行結果

```
2*a+b=24
a-3*b=-2
a*b/b=10
b/a*a=0
a%b%a=2
```

課題 2.4.11. 255 以下の自然数 a を定義し、その数を 2 進数で表示せよ。

ただし、全体を 8 桁で表示させ、a の値を変更しても正しく表示されるプログラムにすること。

```
#include<stdio.h>
int main(void){
  int a;
  a=87;
  printf("%d を 2 進数で表¥示すると",a);
```

```
  printf("です。¥n");
  return 0;
}
```

実行結果

```
87 を 2 進数で表示すると 01010111 です。
```

課題 2.4.12. 浮動小数 (double 型) a,b,c,d を定義し、d に値を定める。d の小数点以下第一位を四捨五入した数 (a)、第二位を四捨五入した数 (b) と、第三位を切り捨てた数 (c) を求め、それぞれを %8.4f で表示するプログラムを作成せよ。

ただし、d の値を変更しても正しく表示されるプログラムにすること。

実行結果

```
d を 18.5236 とします。
18.5236 の小数点以下第一位を四捨五入した数は　19.0000 です。
18.5236 の小数点以下第二位を四捨五入した数は　18.5000 です。
18.5236 の小数点以下第三位を切り捨てた数は　18.52000 です。
```

課題 2.4.13. 定義された 5 つのデータ a1 から a5 の平均値と分散を表示させるプログラムを作成させよ。

ただし、a1 から a5 の値を変更しても正しく表示されるプログラムにすること。

実行結果

```
5 つのデータ 4，-3，5，12，-2 の平均は 3.2 で、分散は 29.36 です。
```

第 3 章

データ入力と乱数

3.1 コンソール入出力

C 言語で標準入力 (キーボード) から文字を入力するには getchar(), gets(), scanf() を使い、標準出力 (ディスプレイ) に文字を出力するには putchar(), puts(), printf() を使う。(どちらも、前者 2 つは使いやすいが汎用性が乏しい。汎用性を考えれば、scanf(), printf() をマスターすることを薦める。)

3.1.1 1 文字の入出力

1 文字のみの入力や出力には getchar(), putchar() を使う。以下の例から解るように、複数文字を入力しても変数に代入される文字は 1 文字だけとなる。

例 **3.1.1.** getchar() を使いキーボードから 1 文字入力する例 1

```
1 #include<stdio.h>
2 void main(void){
3     char ch;
4     ch = getchar();
5     putchar(ch);
6     putchar('¥n');
7 }
```

実行結果 1

```
a   ← キーボードからの入力
a   ← putchar(ch); による出力
```

実行結果 2

```
abc   ← キーボードからの入力
a   ← putchar(ch); による出力
```

例 **3.1.2.** getchar() を使いキーボードから 1 文字入力する例 2

```
1 #include<stdio.h>
2 int main(void){
3     char ch;
4     ch = getchar();
5     putchar(ch);
6     putchar(ch);
7     return 0;
8 }
```

実行結果

```
a   ← キーボードからの入力
aa   ← 5 行目と 6 行目の putchar(ch); による出力、改行がない。
```

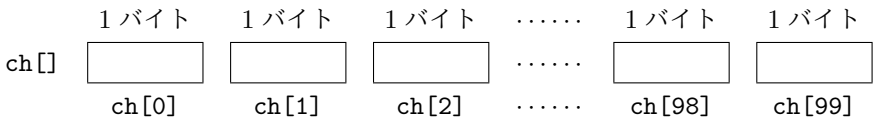
3.1.2　複数文字 (1 行) の入出力

複数文字からなる 1 行の文字列を扱う場合は gets(), puts() を使う。このとき変数として、**配列**を使う。例えば、配列を用いた変数宣言の例として

```
char ch[100];
```

である。これは、1 バイト幅 (char 型) を 100 個用意するという意味である[*1]。

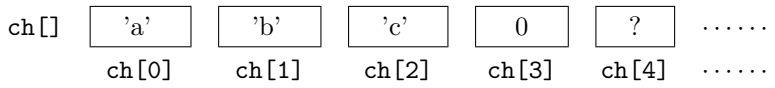
ただし、箱の番号は 0 から 99 となることに注意する。



このとき各箱に 1 文字が入ることになる。たとえば、

```
char ch[100]="abc";
```

とすれば、



となる。このとき、文字列の終わり (終端) に 0 が代入される。(ch[3] が'0' でないことに注意)

また、この 0 は任意に入力することが出来る。

[*1] 配列を用いて文字列を扱う場合、使いたい文字数より多めに宣言をしておく。

例 **3.1.3.** 文字列の終端の扱い

```
 1 #include<stdio.h>
 2 int main(void){
 3     char ch[100]="abcdefg";
 4
 5     printf("%s¥n",ch);
 6     ch[3]=0;
 7     printf("%s¥n",ch);
 8     ch[3]='D';
 9     printf("%s¥n",ch);
10     return 0;
11 }
```

実行結果

```
abcdefg
abc
abcDefg
```

♠ **補足** 文字列の終端は 0 よりも'¥0' を使うことが多い。

♡ **注意!** (何度も出てきたが) 3 行目を分けて、下記のようには出来ない。

```
 3     char ch[100];
 4     ch[100]="abcdefg";
```

例 **3.1.4.** gets() を使いキーボードから 1 行文字列を入力する例 1

```
 1 #include<stdio.h>
 2 int main(void){
 3     char ch[100];
 4     gets(ch);
 5     puts(ch);
 6     puts(ch);
 7     return 0;
 8 }
```

実行結果

```
abc   ← キーボードからの入力 (4 行目)
abc   ← 5 行目の puts(ch); による出力、文字列を出力して改行
abc   ← 6 行目の puts(ch); による出力.(出力後、改行している)
```

実行結果から解るように、puts() は文字列を表示した後、改行してくれる。したがって、puts("") で改行となる。

♠ 補足　ここで、変換指定 %c と %s の違いを思い出す。

```
1 #include<stdio.h>
2 int main(void){
3     char ch[100]="abcdefg";
4     printf("%c ¥n",ch); // NG
5     printf("%s ¥n",ch); // OK
6     printf("%c ¥n",ch[1]); // OK
7     printf("%s ¥n",ch[1]); // NG
8     return 0;
9 }
```

%c は 1 文字を表示し、%s は文字列を表示する。いっけん、7 行目も動作しそうではあるが、実際は動作しない。

例 3.1.5. gets() を使いキーボードから 1 行文字列を入力する例 2

```
1 #include<stdio.h>
2 int main(void){
3     char text[100];
4     gets(text);
5     printf("%s¥n",text);
6     printf("%c¥n",text[2]);
7     return 0;
8 }
```

実行結果

```
abc   ← キーボードからの入力
abc   ← 5 行目の printf(); による出力
c     ← 6 行目の printf(); による出力
```

例 3.1.6. gets() を使いキーボードから 1 行文字列を入力する例 3

```
1 #include<stdio.h>
2 int main(void){
3     char name1[100], name2[100];
4     gets(name1);
5     gets(name2);
6     printf("こんにちは %s %s さん。¥n",name1,name2);
7     return 0;
8 }
```

実行結果

```
Yamada   ← キーボードからの入力 (4 行目)
Taro   ← キーボードからの入力 (5 行目)
こんにちは Yamada Taro さん。
```

3.1.3 数値入力

数値を入力する場合 scanf() を用いる。 scanf() を用いて文字列を入力することもできる。入力するデータによって、%d (int 型) 、%f (float 型) 、%lf (double 型)、%c (char 型)、%s (文字列) などを指定する。

例 **3.1.7.** scanf() を使いキーボードから数値を入力する例

```
 1 #include<stdio.h>
 2 int main(void){
 3     int n;
 4     double d;
 5     scanf("%d",&n);      //代入する変数 n に &が付いていることに注意。
 6     printf("%d¥n",n);
 7     scanf("%lf",&d);     //scanf で double 型を読む場合は %lf にする。
 8     printf("%f¥n",d);
 9     return 0;
10 }
```

実行結果

```
32   ← キーボードからの入力 (5 行目)
32
32.123   ← キーボードからの入力 (7 行目)
32.123000
```

この例の 5 行にあるように scanf() を用いて、int 型の変数に値を入力する場合、 変数 n に&が付いていることに注意する。

例 **3.1.8.** scanf() を使いキーボードから文字列を入力する例

```
1 #include<stdio.h>
2 int main(void){
3     char ch[100];
4     scanf("%s",ch);  //こちらの場合は&が付かない (&ch でない) ことに注意。
5     printf("%s¥n",ch);
6     return 0;
7 }
```

実行結果

```
abc   ← キーボードからの入力 (4 行目)
abc   ← 5 行目の printf(); による出力
```

この例の 4 行にあるように scanf() を用いて、char 型の文字列に値を入力する場合、 変数 n に&が付いていないことに注意する。

♡ **注意!** scanf() 関数は printf() 関数と似ているが、

```
scanf("n の値を入力してください %d",&n);
```

のような使い方はできない。希望の表示方法は

```
printf("n の値を入力してください ");
scanf("%d",&n);
```

とする。

♠ **補足** scanf() 関数は重大なにエラーに対する対処がないため、テストプログラムや入門書を除いてはあまり使われない。

しかし、解り易く便利であり、プログラマと使用者が正しく使用すれば (以下の注意点に気を付ければ) 問題は少ないため、この講義では使用している。(代替 fgets, sscanf)

3.1.4　scanf() の注意点

scanf() 関数は、2 回以上連続で使用する場合、正しく入力しても最後の改行文字はバッファの中に残るため、意図しない動作をするおそれがある（特に %c のとき)。

キーボードから入力された文字は、バッファと呼ばれる領域にいったん読み込まれる。そして、エンターキーが入力されるとバッファをクリアし、再び入力状態になる。このとき、scanf() 関数はエンターキーが入力される前までの文字を読み込み、エンターキーが (改行文字) はバッファに残るため、意図しない動作をするおそれがある。

例 3.1.9. 誤動作の例

```
 1 #include <stdio.h>
 2 int main(void){
 3    int a, b;
 4    char c;
 5    printf("数値 a=");
 6    scanf("%d",&a);
 7    printf("文字 c=");
 8    scanf("%c",&c);
 9    printf("数値 b=");
10    scanf("%d",&b);
11    return 0;
12 }
```

実行結果

```
数値 a=12345
文字=数値 b=67890
```

5, 6 行目の表示, 入力は正しく実行されるが、その後の 7, 8 行目が正しく実行されない。

これは、6 行目の scanf() の際に入力したエンターキー (¥n) がバッファに残り、c に代入されてしまっている。このような動作を防ぐためには %c の前に空白を 1 つ入れて、

```
8 scanf(" %c",&c);
```

とする。か、または、以下のように指定する (こちらはあまり使わないが)。

```
8 scanf("%*c%c",&c);        //文字列なら scanf("%*s%s",ch);
```

とする。連続で入力させたい場合も、空白を入れる。

```
scanf("%c %c %c",&c1,&c2,&c3);
```

3.2 乱数

乱数を用いて、変数に数値を代入する方法がある。乱数を得るにはヘッダファイル stdlib.h を読み込み、rand() 関数を使う。

このとき、rand() は 0 から RAND_MAX(32767) の範囲の整数を発生させる。

例 3.2.1. いろいろな範囲の乱数を生成させる例

(1) a に整数乱数を代入する例

```
1 #include <stdio.h>
2 #include<stdlib.h>
3 int main(void){
4     int a;
5     a=rand();
6     return 0;
7 }
```

(2) 0 から 1 未満の浮動小数乱数を得る方法 (1) の 4, 5 行目を以下に変える

```
4     double b;
5     b=rand()/(RAND_MAX+1.0);
```

これは、0 から 32767 の数を生成し、32768 で割っているので、0 から約 0.99996948242 までの数が 1 つ生成される。もちろん、32768 通りしか得られず、数の間隔は約 0.0000305175 となる。

(3) 1 から 6 の整数乱数を得る方法 (1) の 5 行目を以下に変える

```
5     a=(int)(rand()/(RAND_MAX+1.0)*6)+1;
```

これは、(2) で生成した数を 6 倍した数 (0 から 5.9998168945) の整数部分をキャスト演算子を用いて得たものに 1 を加えている。

クイズ 上記の例の (3) で 1 から 6 の数を生成しているが、なぜ

```
5     a=rand()%6+1;
```

としないのだろうか？

注意しなければならないこととして、rand() 関数は毎回同じ整数を順番に発生させている。異なる整数列を発生させたい場合は、

srand() 関数 (使い方:srand(seed), seed は適当な数)

を用いて rand() 関数で発生させる擬似乱数の発生系列を変更させる。

ただし、srand() 関数の引数 seed に同じ数値を与えると、rand() 関数は同じ整数列繰返しで擬似乱数を発生させる。このことを避けるために、seed として現時刻を使用することが多い。

例 3.2.2. 現時刻を seed として乱数を得る方法

```
 1  #include <stdio.h>
 2  #include <time.h>
 3  #include <stdlib.h>
 4  #include <windows.h>    //Sleep を使うためのヘッダファイル
 5  int main(void){
 6     srand(10);
 7     printf("%d,%d,%d,%d¥n",rand(),rand(),rand(),rand());
 8     srand(10);
 9     printf("%d,%d,%d,%d¥n",rand(),rand(),rand(),rand());
10    srand((unsigned)time(NULL));
11    printf("%d,%d,%d,%d¥n",rand(),rand(),rand(),rand());
12    Sleep(1000);  //すこし時間をずらすため 1000 ミリ秒停める
13    srand((unsigned)time(NULL));
14    printf("%d,%d,%d,%d¥n",rand(),rand(),rand(),rand());
15    return 0;
16 }
```

実行結果

```
8444,10368,10345,30957
8444,10368,10345,30957
3657,19254,31547,15483
14223,9362,9631,15268
```

6 行目で srand(10) を用いて乱数列の発生系列を変更しているが、8 行目で同じ seed を用いて srand() を使っているため、同じ乱数列が表示されている。

10 行目では現時刻を seed として用いているため、先の乱数列とは異なる乱数列となっている。

そのまま現時刻を使うと同じ seed となることがあるので、時間を 1000 ミリ秒だけプログラムを停めて新たな時間を seed として使っている。

3.3 章末問題 III

入力は全てキーボード (標準入力) から入力するものとする。
また、乱数は seed を適当に設定した `srand()` を用いて乱数列を変えること。

課題 3.3.1. 以下の実行結果のように学生番号と氏名の問いに、学生番号と氏名を入力し、入力した文字を画面表示するプログラムを作成せよ。

♠ **補足** 姓と名の間に半角のスペースを入れた場合、姓しか表示されない (ことがある)。

実行結果

```
学生番号を入力してください ： S24M999
氏名を入力してください ： 理大太郎
S24M999 理大太郎
```

課題 3.3.2. 7〜9 桁の自然数 a を入力し、その数 a を 3 桁ずつ区切り、間に , を入れて出力するプログラムを作成せよ。

実行結果

```
自然数 a を入力してください (7〜9 桁) ：3045343
カンマ区切りした a は 3,045,343 です。
```

課題 3.3.3. 乱数を用いて 0 以上の浮動小数 (double 型で小数点以下 3 桁) を生成し、小数点以下第一位を切り捨てた数と、四捨五入した数を表示するプログラムを作成せよ。

実行結果

```
生成された数は 16.534 です。
16.53400 の小数点以下第一位を切り捨てた数は 16 で、四捨五入した数は 17 です。
```

課題 3.3.4. 2 つの自然数 a,b を入力し、a と b の和, 差, 積, 商, 余を表示するプログラムを作成せよ。

実行結果

```
自然数 a を入力してください ：178
自然数 b を入力してください ：84
a と b の和、差、積、商、余はそれぞれ 262，94，14952，2，10 です。
```

課題 3.3.5. 乱数を用いて 1 から 100 までの整数を 10 個生成、表示後、それらの平均を表示させるプログラムを作成せよ。

実行結果

```
10 個の整数 93，31，42，23，56，38，11，76，95，18 の平均は 48.3 です。
```

課題 3.3.6. 自然数と英小文字を入力し、その数の分だけ文字をアルファベット順で後ろの英小文字を表示するプログラムを作成せよ。ただし、アルファベットの列はループするものとし、”a”を 5 ずらすと”f”となり、”y”を 5 ずらすと”d”となるようにせよ。

実行結果

```
自然数 : 5
英小文字 : y
y の 5 文字後は d
```

課題 3.3.7. 英数文字を 5 文字入力し、順番を逆にし表示するプログラムを作成せよ。

実行結果

```
英数文字を 5 文字入力してください。 gS6v7
gS6v7 の順番を逆にすると 7v6Sg です。
```

課題 3.3.8. 実行結果のように、税抜き商品価格を入力すると、消費税 (10%) の金額、合計金額、そして合計金額の 15% のポイントを表示するプログラムを作成せよ。ただし、1 円, 1 ポイント未満は切り捨てとする。

実行結果

```
税抜き価格を入力してください：1280
消費税 128 円、お支払い金額は 1408 円です。
211 ポイントがつきました。
```

課題 3.3.9. 以下の手順で学生番号と OUS-id を作成するプログラムを作成せよ。

まず (大まかな番号を設定するため)、[学生番号 100 の位として]0 から 4 を入力する。

入力が a の場合は乱数を用いて、a00 から a99 までの自然数を、乱数を用いて生成する。

さらに、乱数を用いて、アルファベット小文字 a から z までの文字を 2 文字生成し、OUS-id を作成する。(以下は 155 と g と y が生成された場合)

実行の結果

```
学生番号 100 の位 (0〜4) を入力してください : 1
学生番号は S24M155 で、OUS-id は s24m155gy です。
```

学生番号は S24M で始まり、S24M000 もありうるとする。

課題 3.3.10. まず文字列 (英文字) を入力し、次に (その文字列の文字数より小さい) 自然数 (n) を入力し、文字列の n 番目の文字までを表示するプログラムを作成せよ。

実行結果

```
文字列を入力してください：qwertyuiop
この文字列の文字数より小さい自然数を入力してください：5
文字列 qwertyuiop の 5 番目までは qwert です。
```

第 4 章

制御文

4.1 制御式

制御文は、プログラムの動作を制御する命令であり、選択文 (if , switch)、反復文 (for, while, do-while)、分岐文 (continue, break) などがある。

この選択文や反復文の中には、**制御式**を記述し、動作を制御する。制御式には次の比較演算子 (など) を使う。

4.1.1 比較演算子

比較演算子は、その左辺と右辺の関係が演算子に対して正しい (真) か、誤り (偽) かを判断し、真の場合は 1、偽の場合は 0 の値を返す。

演算子	読み	例	意味
<	小なり	a<b	a は b より小さい
<=	小なりイコール	a<=b	a は b 以下
>	大なり	a>b	a は b より大きい
>=	大なりイコール	a>=b	a は b 以上
==	イコール	a==b	a は b と等しい
!=	ノットイコール	a!=b	a は b と等しくない

算術演算子では、3+5 は、8 であり、2-7 は-5 であった。

比較演算子では、3<5 は、1 であり、2>=7 は 0 である。

例 **4.1.1.** 上記演算子を用いた制御式の例

```
a<1
b>=12
a==b
a!=b
c=(a<=b)
```

例 **4.1.2.** 比較演算子の例

```
#include<stdio.h>
int main(){
   int a;
   a=0;
   printf("%d¥n",a>=3);
   a=5;
   printf("%d¥n",a>=3);
   return 0;
}
```

実行結果

```
0
1
```

4.1.2 真偽値

制御式が条件を満たしていれば、**真 (true)** と言い、満たしていなければ**偽 (false)** と言う。これらを合わせて、**真偽値**と言い、実際は”**真なら 1**”、”**偽なら 0**”の値となる。

例えば、制御式が

制御式：`a==10`

ならば、

真：`a` の値が 10 ならば”真” `a==10` の値は 1

偽：`a` の値が 10 以外ならば”偽” `a==10` の値は 0

となる。

ちなみに、制御式として整数値の真偽値は ”**0 以外なら真**”、”**0 なら偽**”となっている。すなわち、`3` や`-5` は真である。

4.1.3 論理演算子

制御式は論理演算子を用いて組み合わせることができる。

演算子	種類	例	意味
`&&`	論理積 (AND)	`a&&b`	a と b が共に真なら真
`||`	論理和 (OR)	`a||b`	a または b が真のとき真
`!`	否定 (NOT)	`!a`	a が真なら偽、偽なら真

例 **4.1.3.** 論理演算子を用いた例

```
(a>=3 && a<10)   // (a が 3 ≤ a < 10 なら真、a がそれ以外の値なら偽)
(a!=b || b>5)    // (a ≠ b, 5 < b のどちらか成り立てば真、両方不成立なら偽)
(!(a<=b))        // (a > b) と同じ
```

4.1.4 複合代入演算子

よくつかう省略形の演算子として、複合代入演算子がある。

演算子	例	意味
+=	a+=b	a に b を加えた値を a に代入する (a=a+b と同義)
-=	a-=b	a から b を引いた値を a に代入する (a=a-b と同義)
*=	a*=b	a と b を乗じた値を a に代入する (a=a*b と同義)
/=	a/=b	$a \div b$ を a に代入する (a=a/b と同義)
%=	a%=b	$a \div b$ の余りを a に代入する (a=a%b と同義)
>>=	a>>=b	a を b ビット右にシフトした値を a に代入する
<<=	a<<=b	a を b ビット左にシフトした値を a に代入する
&=	a&=b	a と b の論理積を a に代入する
|=	a|=b	a と b の論理和を a に代入する
^=	a^=b	a と b の排他的論理和を a に代入する

"変数 演算子 $=$ 変数" または "変数 演算子 $=$ 定数" の形をしている。

C 言語では

```
a=a+3;    a=a-b;    a=a*b;    a=a/5;
```

などと書くことは稀であり、それぞれ、

```
a+=3;     a-=b;     a*=b;     a/=5;
```

と書くことがほとんど。

例 4.1.4. 複合代入演算子の例

```
 1  #include<stdio.h>
 2  int main(void){
 3     int a=10;
 4     a+=5;
 5     printf("(1) %d¥n",a);
 6     a/=4;
 7     printf("(2) %d¥n",a);
 8     a-=2;
 9     printf("(3) %d¥n",a);
10     a<<=3;
11     printf("(4) %d¥n",a);
12     a*=a;
13     printf("(5) %d¥n",a);
14     return 0;
15  }
```

実行結果

```
(1) 15
(2) 3
(3) 1
(4) 8
(5) 64
```

4 行目は、a=a+5 と同義なので、(1) は 15 となる。

6 行目は、a=a/4 なので、(2) は 3 となる。

8 行目は、a=a-2 なので、(3) は 1 となる。

10 行目は、a=a<<3 なので、(4) は 8 となる。

12 行目は、a=a*a なので、(5) は 8×8 の 64 となる。

4.2　選択文

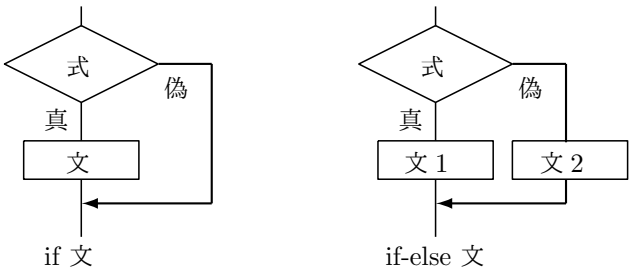
4.2.1　if 文, if-else 文

if 文は制御式の値によって分岐を行う。

書式 以下の文, 文 1, 文 2 は複数の文を { } でまとめた文の場合も可。

```
if(式) 文;
if(式) 文 1 else 文 2 ;
```

それぞれの if 文のフローチャートを表したものが以下である。



例 **4.2.1.** if 文の例 1

1)
```
if(a>10)
   printf("a は 10 より大きい数です。¥n");
```

2)
```
if(a>10)
   printf("a は 10 より大きい数です。¥n");
else
   printf("a は 10 以下の数です。¥n");
```

3)
```
if(a>10 && a%2==0){
   printf("a は 10 より大きい数です。¥n");
   printf("さらに、a は偶数です。¥n");
}
```

例 **4.2.2.** if 文の例 2 (a という制御式は、a が 0 なら偽、それ以外の整数なら真となる。)

```
1 #include <stdio.h>
2 int main(void){
3    int a;
4    scanf("%d",&a);
5     if(a)
6     printf("a は 0 ではありません。¥n");
7     return 0;
8 }
```

♡ **重要!** if 文 (制御文全般) は、後続する 1 つの文を制御する。

例 4.2.3. if 文の例 3

```
a=0;
if(a==1)
printf("a は 1 です。¥n");
printf("a は 0 ではありません。¥n");
```

実行結果

```
a は 0 ではありません。
```

例 4.2.4. if 文の例 4 (複数の文を扱う場合は { } で囲む。)

```
a=0;
if(a==1){
printf("a は 1 です。¥n");
printf("a は 0 ではありません。¥n");
}
```

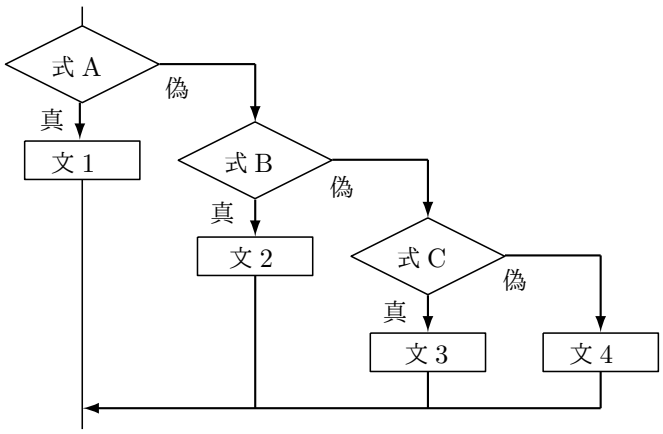
実行結果

このプログラムでは 2 つめの printf 文は動作しない。何も表示されない。

♡ **注意!** 制御文がどこまで制御しているのか解りやすくするためにも字下げを行う。

☆ `if`〜`else` の文には、`else` の後に `if` を付けることもできる。

```
if(式 A){
    文 1;
}
else if(式 B){
    文 2;
}
else if(式 C){
    文 3;
}
else{
    文 4;
}
```



{ , } が増えてくるので付け間違いに注意が必要となる。

例 **4.2.5.** if 文の例 5

1)
```
if(a>10)
     printf("a は 10 より大きい数です。¥n");
else if(a%2==0)
     printf("a は 10 以下で偶数です。¥n");
else
     printf("a は 10 以下で奇数です。¥n");
```

2)
```
if(b%4==0)
     printf("b は 4 で割り切れます。¥n");
else if(b%4==1)
     printf("b を 4 で割ると余りは 1 です。¥n");
else if(b%4==2)
     printf("b を 4 で割ると余りは 2 です。¥n");
else
     printf("b を 4 で割ると余りは 3 です。¥n");
```

例 **4.2.6.** if 文の例 6

例 1.1.5 の買い物問題をプログラムにしてみる。

```
#include<stdio.h>
int main(void){
   int itoda,sayaka;
   printf("ハンバーグ弁当の値段を入力して下さい:);
   scanf("%d,&itoda);
   printf("から揚げ弁当の値段を入力して下さい:);
   scanf("%d,&sayaka);
   if(itoda<sayaka)
      printf("ハンバーグ弁当を買います。¥n");
   else
      printf("から揚げ弁当を買います。¥n");
   return 0;
}
```

4.2.2　条件演算子

if 文と同様の振る舞いをする演算子があるが、あまり使わないので、紹介に止める。

演算子	種類	例	意味
? :	条件演算子	a?b:c	a が真なら b、偽なら c を実行

例 **4.2.7.** 偶数と奇数の数え上げ.(例えば N を配列の要素とするなど)

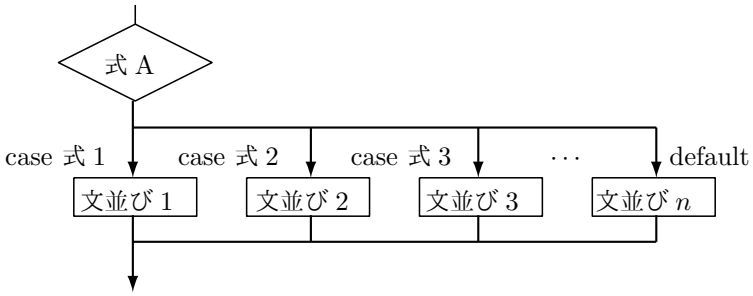
```
(N%2 == 0) ? even++ : odd++;
```

4.2.3 switch 文

if 文は 2 分岐であったが、switch 文は条件によって、いくつかに分岐する。

書式

```
switch(式 A){
case 式 1:
    文並び 1
    break;
case 式 2:
    文並び 2
    break;
case 式 3:
    文並び 3
    break;
  ......
default:
    文並び n
}
```



◇ 式 A には整数値となる変数などを入れる。(`switch(a)` や `switch(i+5)` など。)

◇ 式 1, 式 2, 式 3··· には 式 A の取り得る値を書き、式 A の値と等しい式 1, 式 2, 式 3··· の文並びを実行する。

式 A と等しい値をもつ式 1, 式 2, 式 3··· が無ければ、default の文並びを実行する。

♡ **注意!** `case 式 1`, `case 式 2`, case 式 3, `default` の後はコロン (:)であることに注意。

例 4.2.8. 整数を入力し、その数に応じて分岐する例.

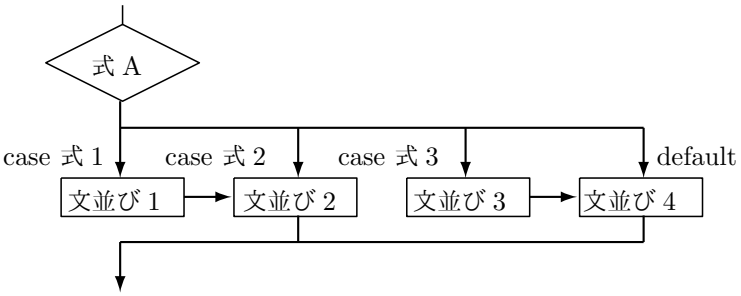
```
scanf("%d", &n);
switch(n){
case 2:
    printf("入力された数は 2 です。¥n);
    break;
case 3:
    printf("入力された数は 3 です。¥n);
    break;
case 6:
    printf("入力された数は 6 です。¥n);
    break;
default:
    printf("入力された数は 2,3,6 以外です。¥n);
}
```

例 4.2.9. 悪い例

```
double n;
scanf("%lf", &n);
switch(n){
case 1.5:
    printf("入力された数は 1.5 です。¥n);
 ......
    break;
```

☆ それぞれの case の文並びの後に break があることに注目。この break が無い場合、後続の case も実行される。

```
switch(式 A){
case 式 1:
    文並び 1
case 式 2:
    文並び 2
    break;
case 式 3:
    文並び 3
default:
    文並び 4
}
```



この場合、式 A が式 1 のとき、文並び 1 と文並び 2 が実行され、式 A が式 2 のとき、文並び 2 が実行される。

これを利用すれば、複数の場合をまとめることが出来る。

例 **4.2.10.** case をまとめた例

```
scanf("%d", &n);
switch(n){
case 2:
case 3:
case 5:
case 7:
    printf("入力された数は 10 以下の素数です。¥n);
    break;
default:
    printf("入力された数は 10 以下の素数ではない。¥n);
}
```

これは、n が、2, 3, 5, 7 のとき、case 7: の次の行に書かれた文並びが実行され、n が 2, 3, 5, 7 以外のときは default: が実行される。

4.3 if 文, switch 文の例

例 **4.3.1.** 課題 1.3.1. を表示させるプログラムの例

```
#include<stdio.h>
int main(void){
  char c;
  printf("家を出発 ¥n");
  printf("晴れているか :");
  c=getchar();
  if(c=='y'||c=='Y'){
     printf("自転車に乗る ¥n");
  }else{
     printf("バスに乗る ¥n");
  }
  printf("大学に到着 ¥n");
  return 0;
}
```

ちなみに、このプログラムでは y，Y 以外はすべて No の扱いとなる。この解消方法は 次の”繰り返し (反復構造)” で学ぶ。

また、課題 1.3.2 を以下のように作成すると、うまく動かない。

```
 1 #include<stdio.h>
 2 int main(){
 3   char c;
 4   printf("スーパーに行く ¥n");
 5   printf("スーパーが休み :");
 6   c=getchar();
 7   if(c=='n'||c=='N'){
 8      printf("りんごの方が安い ");
 9      c=getchar();
10      if(c=='y'||c=='Y'){
11         printf("りんごを買う ¥n");
12      }else{
13         printf("みかんを買う ¥n");
14      }
15   }
16   printf("家に帰る ¥n");
17   return 0;
18 }
```

これは、scanf() のバッファ問題と同じで、改行コードが 9 行目の getchar で読み込まれている。解決方法として、

(1) 9 行目の c=getchar(); を、2 つにして改行コードを読込むダミーを用意する。
(2) 9 行目の c=getchar(); を scanf(" %c",&c); に変える。
(3) 6 行目の c=getchar(); の次の行に fflush(stdin); を入れる。 等々

例題 課題 1.3.3 の内容となるプログラムを作成せよ。

例 **4.3.2.** 課題 1.3.4 は 3 つに分岐するので if 文より switch 文の方が適している。

```
#include<stdio.h>
 int main(){
   int n;
   printf("サイコロを振る ¥n");
   printf("サイコロの目 :");
   scanf("%d",&n);
   switch(n){
   case 1:
   case 2:
   case 3:
      printf("1 つ進む ¥n");
      break;
   case 4:
      printf("1 回休み ¥n");
      break;
   case 5:
   case 6:
      printf("1 つ戻る ¥n");
      break;
   default:
      printf("サイコロの目は 1 から 6 です。¥n");
   }
   return 0;
 }
```

例 **4.3.3.** 文字 (文字コード) にる分岐の例

```
#include<stdio.h>
 int main(){
   char b;
   printf("血液型を半角大文字で入力してください。¥n");
   printf("ただし、AB 型の場合は C と入力してください :");
   scanf("%c",&b);
   switch(b){
   case 'A':
      printf("A 型のあなたは...¥n");
      break;
   case 'B':
      printf("B 型のあなたは...¥n");
      break;
   case 'C':
      printf("AB 型のあなたは...¥n");
      break;
   case 'O':
      printf("O 型のあなたは...¥n");
      break;
   default:
      printf("人類初です！ ¥n");
   }
   return 0;
 }
```

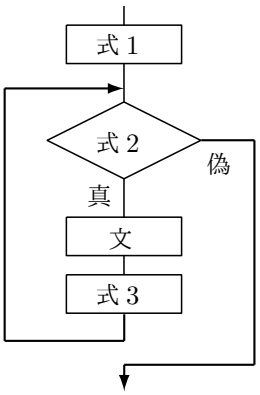
4.4 反復文 1

4.4.1 for 文

for 文は以下の式 2 にかかれた制御式の真偽値が真の間は文を繰り返し実行し、制御式の真偽値が偽になると終わる。

書式

```
for(式 1 ; 式 2 ; 式 3 ) 文;
```



for 文の流れは以下のようになる。

(1) 式 1 を実行する。

(2) 式 2 が真のときは文を実行する。偽なら (4) に行く。

(3) 式 3 を実行し、(2) に戻る。

(4) for 文を終了する。

このとき、式 1, 式 2, 式 3 は省略が可能である。

また、式 2 の制御式によっては、文が 1 度も実行されないこともある。ちなみに、式 1 を初期設定式、式 2 を継続条件式、式 3 を再設定式という。

例 4.4.1. for 文の例

1) `for(i=1;i<=10;i++)`

`i` が 1 から始まり、`i` が 10 まで 1 ずつ増加していく。すなわち、`i` が $1, 2, \ldots, 10$ の 10 回、文を実行する。

2) `for(i=10;i>0;i--)`

これは、`i` が 10 から始まり、$9, 8, \ldots, 1$ まで文を実行する。

3) `for(i=1;i<10;j+=2)`

この式 3 は `j` になっているが、このような使い方もできる。

4) `for(i=1;(i<10&&j<=5);i++)`

この式 2 には `i` と `j` のそれぞれの条件があり、`&&`なので、両方満たしている (`i` が 10 未満でありかつ、`j` が 5 以下) とき、この式 2 が真となり、文を繰り返す。どちらかの条件が偽となれば終了する。

5) `for(;i!=0;)`

これは、`i` が 0 でない間は文を繰り返す。文の中に i の変化が書かれていたり、後述する `break` を文中で用いる。

for 文が良く使われる場合として、"n 回 文を実行する" という**繰り返し (ループ)** 処理がある。上記の例 4.4.1 の 1) は文を 10 回実行するために for 文を用いている。

例 **4.4.2.** 指定回数反復処理 1　(このカッコ { と } は省略可能)

```
for(i=1;i<=3;i++){
    printf("こんにちは ¥n");
}
printf("おわり ¥n");
```

実行結果

```
こんにちは
こんにちは
こんにちは
おわり
```

また、繰り返す間に、回数として変数を使うこともよくある。

例 **4.4.3.** 指定回数反復処理 2 (カッコ { と } を省略)

```
for(i=1;i<=3;i++)
    printf("%d 回目のループ ¥n",i);
printf("おわり ¥n");
```

実行結果

```
1 回目のループ
2 回目のループ
3 回目のループ
おわり
```

再設定式 (式 3) には

`i++` , `++i` , `i+=1` , `i=i+1`

のように 1 増加するようなもの以外の式も記述できる。例えば、

`i--` , `i=i-1` , `i*=2` , `i/=2`

のように 1 ずつ減少や 2 倍増, 半減なども可能である。さらに、式 1, 式 2 で書かれた変数以外で指定することも可能である。

♡ **注意!**　継続条件 (式 2) で注意しなければならないのは、

`i<10` と `i<=10` の違い

である。前者は `i` が 10 となると偽になり、後者は `i` が 10 のとき真である。そのため、for 文が終わったときの `i` の値は前者が 10、後者が 11 である。(参照：例 4.4.5)

for 文も if 文と同様に制御する文は 1 つのみである。したがって、複数の文を制御する場合は {} で囲み、1 つの文としなければならない。

例 4.4.4. 指定回数反復処理 3

```
pow=1;
for(i=1;i<=5;i++){
    pow*=2;
    printf("2 の %d 乗は %d です。¥n",i,pow);
}
```

実行結果

```
2 の 1 乗は  2 です。
2 の 2 乗は  4 です。
2 の 3 乗は  8 です。
2 の 4 乗は 16 です。
2 の 5 乗は 32 です。
```

例 4.4.5. 指定回数反復処理 4

2 の 5 乗を計算するプログラムとして、以下のように入力してみる。(何かおかしい??)

```
pow=1;
for(i=1;i<=5;i++)
    pow*=2;
printf("2 の %d 乗は %d です。¥n",i,pow);
```

実行結果

```
2 の 6 乗は 32 です。
```

この例では i が 6 になると for 文が終わる。そのため、for 文の後に書かれた printf では i の値は 6 と表示される。for 文の終了後に 式 2 で用いた変数の値にも注意する。

式 2 は 例 4.4.1. 3) のように 論理演算子を用いて演算子を組み合わせることが出来る。

例 4.4.6. (i が 100 以下であり、かつ、sum が 1000 以下の場合に繰り返す for 文)

```
sum=0;
for(i=1;(i<=100)&&(sum<1000);i++)
    sum+=i;
```

例 4.4.7. (和が負でない場合に繰り返す for 文)

```
sum=0;
for(i=1;!(sum<0);i++)
    sum+=i;
```

このプログラムでは正の数を足し続けると、いつか負の数になることを意味している。

☆ for 文のなかに for 文を入れることができる。そのときは、式の変数に注意すること。(ループの回数を表す変数として同じものを使わない)

例 **4.4.8.** for 文を 2 つ重ねた例 (二重ループ)

```
 1 #include<stdio.h>
 2 int main(void){
 3    int i,j;
 4    printf("     | 1| 2| 3| 4| 5| 6| 7| 8| 9|¥n");
 5    printf("-----+--+--+--+--+--+--+--+--+--+¥n");
 6    for(i=1; i<=9; i++){
 7        printf("%d の段|",i);
 8        for(j=1;j<=9; j++)
 9            printf("%2d|",i*j);
10        printf("¥n");
11    }
12    return 0;
13 }
```

実行結果

```
     | 1| 2| 3| 4| 5| 6| 7| 8| 9|
-----+--+--+--+--+--+--+--+--+--+
1 の段| 1| 2| 3| 4| 5| 6| 7| 8| 9|
2 の段| 2| 4| 6| 8|10|12|14|16|18|
3 の段| 3| 6| 9|12|15|18|21|24|27|
4 の段| 4| 8|12|16|20|24|28|32|36|
5 の段| 5|10|15|20|25|30|35|40|45|
6 の段| 6|12|18|24|30|36|42|48|54|
7 の段| 7|14|21|28|35|42|49|56|63|
8 の段| 8|16|24|32|40|48|56|64|72|
9 の段| 9|18|27|36|45|54|63|72|81|
```

この例の場合、1 つめの for 文 (7 から 12 行) では変数として i を使い、2 つめの for 文 (9,10 行) の変数として j を使っている。

もし、同じ変数 i のみで行うと

実行結果

```
     | 1| 2| 3| 4| 5| 6| 7| 8| 9|
-----+--+--+--+--+--+--+--+--+--+
1 の段| 1| 4| 9|16|25|36|49|64|81|
```

となって終わる。

4.5 分岐文

4.5.1 break 文

break 文は for 文の中で制御を終了するときに用いられる。

例 4.5.1. キーボードから数値を入力していき 0 が入力されたら break 文を用い入力を終了し、入力された数の合計を表示し、終わるプログラム。

```
 1 #include<stdio.h>
 2 int main(void){
 3     int sum=0;
 4     int n;
 5     for(;;)     //後述の while(1) でもよい。
 6     {
 7         scanf(" %d",&n);
 8         if (n==0)break;
 9         sum+=n;
10     }
11     printf("合計は %d です。¥n",sum);
12     return 0;
13 }
```

実行結果

```
5
-3
12
0           //入力を終わるための 0 の入力
合計は 14 です。
```

6 行目から 10 行目が数値入力をするための for 文が制御する文である。この文の中に、入力された数値が 0 なら break するという if 文がある。

このように反復回数や、明確な終了条件が定まっていない場合 for 文より、while 文や do-while 文を使うことが多い。

例 4.5.1 を 4.6.2 の do-while 文を使った場合、5 行目から 10 行目を

```
 5 do{
 6     scanf(" %d",&n);
 7        sum+=n;
 8 }while(n!=0);
```

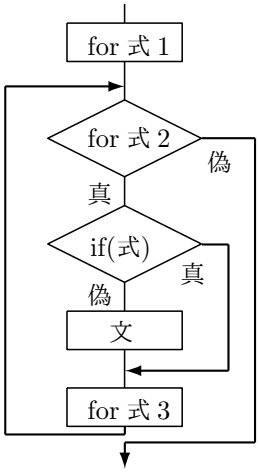
のように書き換えることになる。ただし、この場合は入力された 0 も加えているので、正確に例 4.5.1 の書き換えではない。

4.5.2　continue 文

break 文は switch 文や for 文の中で用いられ、制御を終了させるために使用されたが、continue 文は、for 文などでその回の制御をパスするために使用される。

例 4.5.2. for 文の中に continue を含む if 文を入れた場合の例

```
1 #include<stdio.h>
2 int main(void){
3   int i;
4   for(i=1;i<=30;i++){
5      if(i%2!=0 || i%3!=0) continue;
6      printf("%2d は 6 の倍数 ¥n",i);
7    }
8    return 0;
9 }
```



この例では、5 行目で i が 2 で割れない、または 3 で割れない場合、for 文の制御文の残りをパスし、for 文の 式 3 へ飛ぶ。

例 4.5.3. continue と break の違いの例

```
10  for(i=1;i<10;i++){
11     if(i%4==0) break;
12      printf("%d は 4 で割り切れないよ ¥n",i);
13  }
```

実行結果

```
1 は 4 で割り切れないよ
2 は 4 で割り切れないよ
3 は 4 で割り切れないよ
```

11 行目を `if(i%4==0) continue;` に変えた場合の実行結果

```
1 は 4 で割り切れないよ
2 は 4 で割り切れないよ
3 は 4 で割り切れないよ
5 は 4 で割り切れないよ
6 は 4 で割り切れないよ
7 は 4 で割り切れないよ
9 は 4 で割り切れないよ
```

♠ **補足**　break 文、continue 文はプログラムの繰り返しを中断し、ループから抜けるために便利な機能であるが、しっかり理解して使わなければ思わぬ動作をする。

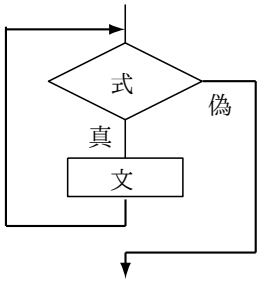
4.6 反復文 2

4.6.1 while 文

while 文は for 文と同様に、制御式の真偽値が真の間は繰り返し、偽になると終わる。

書式

```
while(式) 文;
```



for 文と while 文は似ているが、一般的に for 文は繰り返し回数 (反復条件) が解っている場合に使用し、while 文は繰り返し回数が解らない、指定しない場合に用いられる。

また、while 文は for 文の式 2 のみを書いたものと似ている。

```
while(i<=10){ }
```

は、

```
for(;i<=10;){ }
```

と同じである。

逆に、

```
for(i=1;i<=10;i++){
  文 ;
}
```

と書いたものは while 文を用いて

```
i=1;
while(i<=10){
  文 ;
  i++;
}
```

と書くことが出来る。

例 4.6.1. while 文の例

1) `while(i<=1000)`
2) `while(i!=0)`
3) `while(i>=j)`
4) `while(1)` `//break` 文で抜ける。(例 4.5.1 参照)

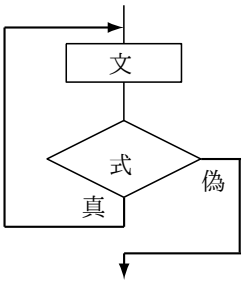
♡ **注意!** while 文では制御式によっては、制御文が 1 度も実行されないことがある。制御式によらず、制御文を実行したい場合、次を用いる。

4.6.2 do-while 文

do-while 文は while 文と同様に、制御式が真の間は繰り返し、偽になると終わる。

書式

```
do 文 while(式);
```



while 文と do-while 文の違いは、式 (制御式) を判別し
文を行うか、文を行ってから式を判別するかであり、
do-while では式の真偽に関らず、まず文が実行される。

♡ **注意!** while() の後に ; を忘れずに。

例 4.6.2. 文字列の中で'g' が出てくるまで表示するプログラム

```
 #include<stdio.h>
1 int main(void){
2    int i=0;
3    char m[50]="abcdefghijk";
4    do {
5       printf("%c",m[i]);
6    } while(m[i++]!='g');
7    printf("¥n");
8    return 0;
9 }
```

クイズ では 例 4.6.2 において do-while を使わず、while で書くにはどうすれば良いか。

ちなみに、

```
4    while(m[i++]!='g'){
5       printf("%c",m[i]);
6    }
```

としただけでは、

実行結果

```
bcdefg
```

となり、4 行目を

```
4    while(m[++i]!='g'){
```

とすると

実行結果

```
abcdef
```

となる。

4.7 for 文、while 文の例

例 **4.7.1.** 1〜n までの和を求めるプログラムを、例 1.2.1 のアルゴリズムを参考に考える。
なお、プログラムは for 文を用いるものと、while 文を用いるものを考える。

プログラム：($n = 4$ のとき、for 文を用いた例)

```
#include<stdio.h>
int main(void){
   int sum=0,i,n=4;
   for(i=1;i<=n;i++){
      sum+=i;
   }
   printf("1 から %d までの和は %d です。¥n",n,sum);
   return 0;
}
```

プログラム：($n = 4$ のとき、while 文を用いた例)

```
#include<stdio.h>
int main(void){
   int sum=0,i=1,n=4;
   while(i<=n){
      sum+=i;
      i++;
   }
   printf("1 から %d までの和は %d です。¥n",n,sum);
   return 0;
}
```

例 **4.7.2.** 入力された数の約数を表示するプログラムを考える。

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
   int i,n;
   srand(10);
   printf("10000 以上の自然数を入力してください：");
   scanf("%d",&n);
   printf("%d の約数は 1",n);
   for(i=2;i<=n;i++)
      if(n%i==0) printf(",%d",i);
   printf("です。¥n");
   return 0;
}
```

実行結果

```
10000 以上の自然数を入力してください：12345
12345 の約数は 1,3,5,15,823,2469,4115,12345 です。
```

例 **4.7.3.** 1 から 100 までの乱数を 100 個生成し、それらの数の平均を表示させるプログラムを考える (参考 課題 3.3.5)。なお、生成された数は 10 個ずつで改行する。

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
   int i,j,sum=0,n=100;
   srand(20);
   for(i=1;i<=n;i++){
     j=(int)(rand()/(RAND_MAX+1.0)*100+1);
     sum+=j;
     printf("%3d,",j);
     if(i%10==0) printf("¥n");
   }
   printf("これらの平均は %4.2f です。¥n",sum/100.0);
   return 0;
}
```

実行結果

```
  1, 80, 56, 77, 57, 76, 16, 20, 40, 19,
  9, 89, 55, 11, 18,  7, 65,  5, 21, 81,
 61, 43, 83,  8, 48, 80, 89, 13, 40, 43,
 91, 92, 53, 27, 34, 84, 98, 74, 18, 54,
 40, 55, 76, 15, 47, 92, 17, 67, 85, 30,
 17, 83, 82, 17,  9, 10, 18, 79, 46, 31,
 70, 66, 93, 49, 63, 51, 41, 31, 52, 46,
 58, 87, 99, 43,  3, 13, 57, 11, 15, 73,
 32, 88, 85, 56, 31, 80, 41, 73, 75,  7,
 44, 40, 75, 20, 32,  6, 48, 18,  1, 66,
これらの平均は 47.91 です。
```

例 **4.7.4.** 1 から 1000 までの乱数を 100 個生成し、それらの数の最大値と最小値を表示させるプログラムを考える。

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
   int i,j,n=100;
   int max=-1*(1<<30);
   int min=1<<30;
   srand(20);
   for(i=1;i<=n;i++){
     j=(int)(rand()/(RAND_MAX+1.0)*1000+1);
     if(j>max)
        max=j;
     if(j<min)
        min=j;
   }
   printf("これらの最大値は %4d で、最小値は %4d です。¥n",max,min);
   return 0;
}
```

4.8 章末問題 IV

課題 4.8.1. 整数を入力し、入力された数が正の数か、負の数か、0 か表示するプログラムを作成せよ。

実行結果 1

```
整数を入力してください。：-21
入力された数は負の数です。
```

実行結果 2

```
整数を入力してください。：0
入力された数は 0 です。
```

課題 4.8.2. テストの点数を入力し、評価 (S,A,B,C,D) を表示するプログラムを、if 文を用いず switch 文を用いて作成せよ。ただし、S は 90 点から 100 点、A は 80 点から 89 点、B は 70 点から 79 点、C は 60 点から 69 点、D は 60 点未満とする。(ヒント: 商)

実行結果 1

```
テストの点数を入力してください。：85
評価は A です。
```

実行結果 2

```
テストの点数を入力してください。：59
評価は D です。
```

課題 4.8.3. 4 桁の自然数 (y) を入力し、y 年がうるう年かどうか判断するプログラムを作成せよ。ただし、うるう年か平年かは、以下の規則に従う。(グレゴリオ暦)

(i) 西暦年が 4 で割り切れる年はうるう年。

(ii) ただし、西暦年が 100 で割り切れる年は平年。

(iii) ただし、西暦年が 400 で割り切れる年はうるう年。

実行結果 1

```
西暦を入力してください。：2100
西暦 2100 年は平年です。
```

実行結果 2

```
西暦を入力してください。：2800
西暦 2800 年はうるう年です。
```

課題 4.8.4. 2 桁の自然数 (n) を入力し、n は 2,3,5,7 の倍数かそれぞれ、表示するプログラムを作成せよ。

実行結果

```
2 桁の自然数を入力してください。：40
40 は 2 の倍数です。
40 は 3 の倍数ではありません。
40 は 5 の倍数です。
40 は 7 の倍数ではありません。
```

課題 4.8.5. 標準入力から 3 文字入力し、大文字は小文字に、小文字は大文字に変換し、数字や記号はそのまま表示するプログラムを作成せよ。ただし、入力は 3 文字であると仮定してよい。

実行結果

```
英数文字を 3 文字入力してください：aG8
Ag8
```

課題 4.8.6. (1) 例 4.2.5 の 2) に`#include<stdio.h>`, 変数宣言, `scanf(" %d",&b);` などを付け加え、if 文を switch 文に変えたプログラムを作成せよ。
(2) 例 4.3.1 に`#include<stdio.h>`や変数宣言などを付け加え、switch 文を、if 文に変えたプログラムを作成せよ。

課題 4.8.7. (電卓プログラム) int 型変数 a,b と、char 型変数 op を用意する。a, op, b の値をこの順に入力し、op が +(加算), -(減算), *(乗算), /(除算), %(剰余) のいずれかであればその計算をして表示するプログラムを switch 文を用いて作成せよ。

実行結果

```
整数を 1 つ入力してください (a=):  26
演算子 (+,-,*,/,%) を 1 つ入力してください:  %
整数をもう 1 つ入力してください (b=):  4
 26 % 4 の結果は 2 です。
```

実行結果

```
整数を 1 つ入力してください (a=):  6
演算子 (+,-,*,/,%) を 1 つ入力してください:  $
整数をもう 1 つ入力してください (b=):  8
 そのような計算は出来ません。
```

課題 4.8.8. 例 4.7.1 を参考に 1 から n までの奇数の和を求めるプログラムを作成せよ。

なお、n は (`int n=10;` のように) 自由に自然数を指定して良い。

課題 4.8.9. 例 4.7.1 を参考に $\displaystyle\sum_{i=1}^{10} i^2$ を求めるプログラムを作成せよ。

課題 4.8.10. フィボナッチ数列の第 n 項 ($n \geqq 2$) を表示するプログラムを作成せよ。

なお、n は (`int n=10;` のように) 自由に 2 以上の自然数を指定して良い。

課題 4.8.11. 10000 以上の自然数を入力し、その数が素数か否か判断するプログラムを作成せよ。

実行結果

```
10000 以上の自然数を入力してください：12347
12347 は素数です。
```

課題 4.8.12. 英小文字を 4 つ入力し、暗号キーとして自然数 (整数) を 1 つ入力すると、その数の分だけ文字をアルファベット順で後ろにずらした暗号文の表示を繰り返し、平文の先頭に”.”が入力されると終了するプログラムを作成せよ。

たとえば、”a”を 5 ずらすと”f”となる。またアルファベットの列はループするものとし、”y”を 5 ずらすと”d”となるようにせよ。

実行結果

```
平文 : math
暗号キー : 15
暗号文は "bpiw"です。
平文 : qwer
暗号キー : 12
暗号文は "ciqd"
平文 : .fin
終わります。
```

課題 4.8.13. (コラッツの予想は表現とメディアの数理のプリント、またはネット検索)

自然数を入力し、コラッツの予想が正しいか、1 になるまでの過程を表示するプログラムを作成せよ。ただし、表示する数は 10 個ずつで改行を入れること。

実行結果

```
2 以上の自然数を入力してください : 7
   7,  22,  11,  34,  17,  52,  26,  13,  40,   20,
  10,   5,  16,   8,   4,   2,   1,
```

課題 4.8.14. （ユークリッドの互除法）

2 つの自然数を入力し、その 2 つの数の最大公約数を求めるプログラムを作成せよ。

実行結果

```
2 以上の自然数を入力してください。: 14
2 以上の自然数を入力してください。: 35
14 と 35 の最大公約数は 7 です。
```

課題 4.8.15. 英数文字を入力し、大文字は小文字に、小文字は大文字に変換し、数字はそのまま表示するプログラムを作成せよ。ただし、先頭に . が入力されたら終了する。

実行結果

```
文字列を入力してください : aB1
大文字小文字変換したものは Ab1 です。
文字列を入力してください : a1fg4Rkju90
大文字小文字変換したものは A1FG4rKJU90 です。
文字列を入力してください : .
終わります。
```

課題 4.8.16. 標準入力から 10 個の整数を入力し、それら 10 個の数の平均と最大値、最小値を表示するプログラムを作成せよ。

実行結果

```
10 個の整数値を入力してください。
15
-4
0
3
-4
20
-14
15
-6
7
10 個のデータの平均は 3.2 で、最大値は 20, 最小値は-14 です。
```

課題 4.8.17. 自然数 (n) を入力し、2 から数えて、n 番目の素数を表示するプログラムを作成せよ。

実行結果

```
2 以上の自然数を入力してください。: 30
2 から数えて 30 番目の素数は 113 です。
```

第 5 章

配列と文字列処理

5.1 配列

5.1.1 1 次元配列

同種のデータ型を連続してメモリ上に確保したものを**配列**という。(文字列と同様)

例 `int a[10];`

`double b[10];`

変数名の後の[ ]の中の数はサイズであり、サイズが 10 の場合、添字は 0 から 9 となる。(文字列を扱う場合に定義した配列と同じである。)

宣言された配列は変数と同様に代入や演算を行うことができる。例えば、例で宣言された配列 a に対して、

`a[0]=1;a[1]=1;a[2]=a[0]+a[1];a[3]=a[1]+a[2];`

のように、a[0],...,a[3] に値を代入することが出来る。

例 5.1.1. フィボナッチ数列を初項から第 20 項まで表示させるプログラム。

```
#include<stdio.h>
int main(void){
  int a[100], i;
  a[0]=1;
  a[1]=1;
  for(i=2;i<20;i++)
    a[i]=a[i-1]+a[i-2];
  for(i=0;i<20;i++)
    printf("a[%2d]=%4d¥n",i,a[i]);
  return 0;
}
```

注) 初項は a[0] で、第 20 項が a[19] であることに注意すること。

例 **5.1.2.** 標準入力から整数型のデータを 10 個入力し、配列に格納する例

```
#include<stdio.h>
int main(void){
  int data[100],i;
  for(i=0;i<10;i++){
    scanf(" %d",&data[i]);
  }
  return 0;
}
```

(慣れるまで) 配列のサイズは、扱うデータより少し多めに宣言しておくのが良い。

5.1.2　2 次元配列

配列は 1 次元のものだけでなく、2 次元も扱える。

例えば、2 次元配列

```
double a[5][4];
```

は数学の行列と思うことも出来る。宣言されたものを並べると

```
a[0][0]  a[0][1]  a[0][2]  a[0][3]
a[1][0]  a[1][1]  a[1][2]  a[1][3]
a[2][0]  a[2][1]  a[2][2]  a[2][3]
a[3][0]  a[3][1]  a[3][2]  a[3][3]
a[4][0]  a[4][1]  a[4][2]  a[4][3]
```

となる。

例 **5.1.3.** 成分が整数の 2 次正方行列を入力し、行列式を表示するプログラム。

```
#include<stdio.h>
int main(void){
  int i,j,det;
  int m[2][2];
  for(i=0;i<2;i++){
    for(j=0;j<2;j++){
      printf("%d 行 %d 列成分を入力してください:",i+1,j+1);
      scanf(" %d",&m[i][j]);
    }
  }
  det=m[0][0]*m[1][1]-m[1][0]*m[0][1];
  printf("行列式の値は %d です。¥n", det);
  return 0;
}
```

5.1.3 初期化

変数を宣言するときに、最初の値を設定することができる。これを**初期化**という。

配列を扱う場合、毎回 scanf() でデータを読み込むことは手間がかかるため、(特にプログラムの作成中は) 配列を初期化して使用することが有用である。

例
```
3  int a=20;
4  double b=3.14;
5  char c='A';
6  int d[3]={1,2,3};
7  char f[]={'A','B','C'};
8  char g[10]="abcdef";
9  int h[3]={0,1,2,3};
10 char i[3]="abcde";
```

9,10 はエラーとなる例であり、どちらも宣言しているサイズより多くのデータを入れようとしている。

また、初期化をしなかった場合、不定値となる。(宣言後に代入もせず) 不定値のまま変数値を使おうとすると、通常は警告が表示される。この講義 (実習) で使用する bcpad では警告は出ない (さらに int 型なら値が 1 となる)。

```
#include<stdio.h>
int main(void){
  int i;
  printf("%d¥n", i);
  return 0;
}
```

文字列を扱う場合、初期化の時には文字の代入のような作業を行うことができる。しかし、変数宣言後に

```
g[10]="abcdef";
g[]="abcdef";
```

のようなことは出来ない。

例 5.1.4. 初期化した 10 文字の文字列を逆から表示する例

```
#include<stdio.h>
int main(void){
  int i;
  char data[20]="tcsygogmia";
  for (i=9; i>=0; i--)
    printf("%c,",data[i]);
  printf("¥n");
  return 0;
}
```

5.1.4　文字の辞書式ソート

例 5.1.5. 例 5.1.4. において、初期化で与えられた 10 文字を辞書順に並べ替え、表示するプログラム文字の例

```
 1 #include<stdio.h>
 2 int main(void){
 3   int i;
 4   char temp,data[20]="tcsygogmia";
 5
 6   for(i=0;i<10;i++)
 7     printf(" %c,",data[i]);
 8   printf("¥n");
 9   i=0;
10
11   while(i<9){
12     i++;
13     if(data[i-1]>data[i]){
14       temp=data[i-1];
15       data[i-1]=data[i];
16       data[i]=temp;
17       i=0;
18     }
19   }
20
21   for(i=0;i<10;i++)
22     printf(" %c,",data[i]);
23   printf("¥n");
24   return 0;
25 }
```

実行結果

```
t, c, s, y, g, o, g, m, i, a,
a, c, g, g, i, m, o, s, t, y,
```

ソートについては、いろいろな方法があるので各自で調べてみて欲しい (1.2.3 ソートのアルゴリズムも参照)。

♠ **補足**　アルファベット大文字 (文字コード 65〜97) がある場合、小文字 (文字コード 97〜129) を間に入るように番号付けを変えればよい。

例えば、`(data[i]/32)%2` は大文字なら 0、小文字なら 1 となり、`data[i]%32` は A または a が 1、B または b が 2 ..... となっていく。

これを利用し、`(data[i]/32)%2+(data[i]%32)*2)` を計算してみると...。

5.2 文字列処理関数

C 言語では文字列の複写、連結、比較等はすべて関数で行う。その際、

```
#include<string.h>
```

が必要となる。

文字列処理関数	
strcpy(ss, st)	文字列 ss に st をコピーする
strcat(ss, st)	文字列 ss の後ろに st を連結する
strcmp(st1, st2)	文字列 st1 と st2 を比較する 文字数によらず、文字の大小関係で結果が決まる st1 > st2 なら戻り値は正の数 st1 < st2 なら戻り値は負の数 st1=st2 なら戻り値は 0
strlen(st)	文字列の長さを返す
strncpy(ss, st, n)	文字列 ss に st の先頭から n 文字をコピーする
strncat(ss, st, n)	文字列 ss の後ろに st の先頭から n 文字を結合する
strncmp(st1, st2, n)	文字列 st1 と st2 の先頭から n 文字を比較する

☆ strcmp(ss, st) の動作については以下の例 5.2.2 で説明する。

使用例としては、以下のようになる。

例 **5.2.1.** 文字列処理関数の例

```
 1 #include<stdio.h>
 2 #include<string.h>
 3 int main(void){
 4    char a[]="abc";
 5    char b[]="def";
 6    printf("文字列¥"%s¥"の長さは %d です。¥n", a, strlen(a));
 7    strcpy(a,b);
 8    printf("文字列¥"%s¥"の長さは %d です。¥n", a, strlen(a));
 9    strcat(a,b);
10    printf("文字列¥"%s¥"の長さは %d です。¥n", a, strlen(a));
11    return 0;
12 }
```

実行結果

```
文字列"abc"の長さは 3 です。
文字列"def"の長さは 3 です。
文字列"defdef"の長さは 6 です。
```

例 5.2.2. strcmp の実行例 (ただし、処理系により値は異なり、$-1, 0, 1$ のみ表示される場合がある。)

```
 1 #include<stdio.h>
 2 #include<string.h>
 3 int main(void){
 4    printf("1) %d¥n", strcmp("ABC","ABC"));
 5    printf("2) %d¥n", strcmp("ABC","ABD"));
 6    printf("3) %d¥n", strcmp("ABC","AAAA"));
 7    printf("4) %d¥n", strcmp("ABC","AB"));
 8    printf("5) %d¥n", strcmp("AB","CC"));
 9    printf("6) %d¥n", strcmp("AB","ab"));
10    return 0;
11 }
```

実行結果

```
1) 0
2) -1
3) 1
4) 67
5) -2
6) -32
```

(考え方) 先頭から比較し、異なるところで比較した差を返す。その時の、各文字の値は

A	B	C	D	E	F	...	Z	a	b	c	d	e	...	z
65	66	67	68	69	70	...	90	97	98	99	100	101	...	122

である。

1) 2 つの文字列が完全に一致している場合、`strcmp()` の値は 0 となる。

2) 先頭から比べていき、初めて異なる文字 ”C” と”D” の差 `-1` を返している。

4) ”AB”までは同じで、一方は”C”で他方は””なので、”C”の数 67 を返している。

6) ”A”と”a”の差、`-32` を返している。

♠ **補足** `strncpy(ss,st+k,n);` 文字列 `ss` に `st` の k 番 ($k+1$ 文字目) から n 文字をコピーする。

例 `strncpy(ss,st+3,5);`

`st` の 3 番 (0,1,2,3 なので文字では 4 文字目) から 5 文字を `ss` にコピーする。

すなわち `a[]` が

`"abcdefghijklmn"`

のとき、`strncpy(b,a+3,5);` とすると `b[]` は

`"defgh"`

となる。

5.3 章末問題 V

課題 5.3.1. 標準入力から西暦と月日を入力し、その年の何日目か表示するプログラムを作成せよ。なお、うるう年も正しく考慮すること。

実行結果

```
西暦を入力してください ： 2000
何月ですか ： 3
何日ですか ： 1
2000 年 3 月 1 日は 2000 年の 61 日目です。
```

課題 5.3.2. 乱数を用いて 0.00 から 9.99 までの浮動小数を 10 個生成し、生成された数の最小値、最大値、中央値、第 1 四分位数と第 3 四分位数を表示するプログラムを作成せよ。ただし、乱数は時刻を seed とした srand() を使用した後、生成させること。

実行結果

```
乱数で得られた浮動小数は
1.72，5.43，4.14，2.42，7.44，8.64，3.78，4.48，2.05，1.39，
であり、これらを小さい順に並べたものは
1.39，1.72，2.05，2.42，3.78，4.14，4.48，5.43，7.44，8.64，
である。これより、最小値は 1.39、最大値は 8.64 であり、
中央値、第 1 四分位数と第 3 四分位数はそれぞれ 3.96,2.05,5.43 である。
```

課題 5.3.3. 標準入力から 2 つの文字列を入力し、先頭から比べて始めて異なる文字を表示するプログラムを作成せよ。ただし、必ず異なる文字列が入力されるものとしてよい。

実行結果

```
英文字を入力してください: Uchida
英文字を入力してください: Uchinashi
文字列 Uchida と Uchinashi で最初に異なる文字は 5 文字目の d と n です。
```

課題 5.3.4. 標準入力から整数を 10 個入力し、入力された整数の平均値と標準偏差および、各整数の偏差値を表示するプログラムを作成せよ。ただし、平方根はヘッダファイルとして、`math.h` を読み込み、`sqrt()` で求めることができる。($\sqrt{d}$ は `sqrt(d)` である。)

ちなみに、N 個のデータ x_i $(i = 1, \ldots, N)$ に対して、平均 μ、標準偏差 σ、x_i の偏差値 T_i はそれぞれ

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i,\ \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} {x_i}^2 - \mu^2},\ T_i = \frac{10(x_i-\mu)}{\sigma} + 50$$

である。

課題 5.3.5. 文字を 10〜20 文字入力し、前後を 2 分割して入れ替えて表示するプログラムを作成せよ。ただし、奇数文字の場合は前半を後半より 1 文字多く別けること。

実行結果

```
文字列を入力してください：a1fg4Rkju90
文字列の前半後半を入れ替えた文字列は kju90a1fg4R です。
```

課題 5.3.6. 以下の 2 段階で、浮動小数のソートを行うプログラムを考える。(A),(B) の順でプログラムを完成させよ。ただし、数値の表示の桁数は実行結果と同じものとし、乱数は時刻を seed とした srand() を使用した後、生成させること。

(A) 01 番から 10 番までの浮動小数 (0.00〜9.99) を乱数を用いて (10 個) 生成し、以下の実行結果 (A) のように表示するプログラムを作成せよ。

(B) (A) で表示されているそれぞれの浮動小数を小さい順に並び変えて、実行結果 (B) のように表示されるようなプログラムを作成せよ。

ただし、実行結果 (A) の後に (下に) 続いて実行結果 (B) が表示されるようにすること。

実行結果 (A)

```
10 個のデータは以下の通り
01 番 4.72
02 番 7.55
03 番 3.75
04 番 5.92
05 番 0.34
06 番 1.22
07 番 8.18
08 番 0.61
09 番 9.18
10 番 6.70
```

実行結果 (B)

```
小さい順にソートすると以下の通り
05 番 0.34
08 番 0.61
06 番 1.22
03 番 3.75
01 番 4.72
04 番 5.92
10 番 6.70
02 番 7.55
07 番 8.18
09 番 9.18
```

課題 5.3.7. 3 つの文字列 (アルファベット小文字のみ) を入力し、それらを辞書式に並べ替えて表示するプログラムを作成せよ。

実行結果

```
1 つ目の文字列を入力してください。: okayama
2 つ目の文字列を入力してください。: hiroshima
3 つ目の文字列を入力してください。: okinawa
3 つの文字列を辞書式に並べ変えたものは、
hiroshima
okayama
okinawa
です。
```

課題 5.3.8. 自然数 ($1 \leqq n \leqq 10^7$) を 1 つ入力し、n の先頭または、末尾、もしくは各桁の間に 1 桁の数を挿入する操作を 1 回おこなうことによって出来る数は何通りあるか表示するプログラムを作成せよ。ただし、先頭に 0 を挿入は数えない。

実行結果

```
自然数 (1〜10000000) を入力してください。: 8
18 通りあります。
```

求める数は、$18, 28, 38, 48, 58, 68, 78, 88, 98, 80, 81, 82, 83, 84, 85, 86, 87, 89$ の 18 通りである。88 を 2 回数えないことと、08 は含まないことに注意。

課題 5.3.9. 台形の上底、下底、高さを入力し、この台形の面積を表示するプログラムを作成せよ。ただし、存在しない台形の場合を除外する必要はない。

実行結果

```
上底を入力してください。: 3
下底を入力してください。: 4
高さを入力してください。: 5
台形の面積は 17.50000 です。
```

小数点以下が出る場合に注意。小数点以下の桁数 (0 の数) は気にしなくて良い。

課題 5.3.10. 50 文字以下の文字列を入力し、その文字列の中にある数字だけを足した合計を表示するプログラムを作成せよ。

実行結果

```
50 文字以下の文字列を入力してください。: a6513gar4q6uq07tgae
文字列の中の数字の合計は 32 です。
```

課題 5.3.11. 入力する数値の数を入力し、その個数の自然数を入力し、それら合計を表示するプログラムを作成せよ。ただし、数字が無い場合 0 と表示するようにすること。

実行結果

```
入力するデータ数を入力してください。: 4
7
14
0
11
合計は 32 です。
```

課題 5.3.12. 3 以上の自然数 (n) を入力し、フィボナッチ数列の第 0 項から第 n 項 F_n まで表示するプログラムを作成せよ。ただし、$F_0 = 1, F_1 = 1, F_{n+2} = F_{n+1} + F_n$ とする。

実行結果

```
フィボナッチ数列の第何項まで表示しますか ： 10
1,1,2,3,5,8,13,21,34,55,89,
```

課題 5.3.13. 以下のプログラムにおいて、代入 (宣言) された int 型変数 a、double 型変数 b、文字変数 c、文字列変数 d を変換指定を用いて実行結果のように表示するプログラムを完成させよ。

なお、変数の宣言は、2.3.1 で説明予定。

```
#include <stdio.h>
int main(void){
   int a=5;
   double b=3.14159;
   char c='M';
   char * d="abcd";
   printf(                                );
   return 0;
}
```

実行結果

```
表示するものはそれぞれ、5,3.14159,M,abcd です。
```

課題 5.3.14. 素数 (p) を入力し、p の平方根を小数点以下 3 桁まで表示するプログラムを作成せよ。ただし、小数点以下 4 桁目を四捨五入して 3 桁とすること。

なお、この講義内で習った範囲で答えること。

実行結果

```
素数 p を入力してください ： 13
13 の平方根の近似値は 3.606 です。
```

課題 5.3.15. 11〜20 の自然数 (n) と、5〜10 の自然数 (r) を入力し、二項係数 ${}_nC_r$ を表示するプログラムを作成せよ。

なお、関数は main 関数のみで、再帰関数などは使用禁止。

実行結果

```
11〜20 の自然数を入力してください ： 20
5〜10 の自然数を入力してください ： 10
二項係数 20C10 は 184756 です。
```

課題 5.3.16. −10〜10 の整数を 4 個入力し、順に a, b, c, d とするとき、行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ の 3 乗を

表示するプログラムを作成せよ。

実行結果

```
-10〜10 の自然数をカンマ区切りで 4 個入力してください ： 3,-2,5,12
( 3 -2)
( 5 12) の 3 乗は
( 27 -8)
( 125 1728) です。
```

課題 5.3.17. 10000 以下の素数 p を入力し、$p = a^2 + b^2$ を満たす自然数 a, b の組が存在すれば、(実行結果のように)a, b を、存在しなければ “無し” と表示するプログラムを作成せよ。ただし、$a \leqq b$ とする。

実行結果

```
10000 以下の素数を入力してください ： 97
 97 は (4 の 2 乗)+(9 の 2 乗) です。
```

課題 5.3.18. 4〜9 の整数 N を入力する。次に、1 から 9 までの数から、重複を許し N 個入力し、それらの数から 4 個選び、4 桁の数を作るとき全部で何通りあるか表示するプログラムを作成せよ。

実行結果

```
N を入力してください (4〜9) ： 6
1 から 9 までの数を 6 個入力してください。
 4
 3
 1
 8
 4
 3
 4,3,1,8,4,3 から 4 つ選び出来る 4 桁の数は 102 個です。
```

課題 5.3.19. (最長 20 文字の) 文字列を入力し、文字列の中のアルファベット大文字、小文字、数字、その他の文字の数を表示させるプログラムを作成せよ。

実行結果

```
文字列を入力してください。： aHriA462$kh1b#R(68uT
入力された文字列 aHriA462$kh1b#R(68uT の中に、
アルファベット大文字は 4 文字、小文字は 7 文字あります。
数字は 6 文字あります。
上記以外は 3 文字あります。
```

課題 5.3.20. (最長 50 文字の) 文字列を入力し、入力された文字列の長さの表示を、先頭文字にドット (.) が入力されるまで繰返すプログラムを作成せよ。

実行結果

```
文字列を入力してください。: b38khuR
入力された文字列は 7 文字です。
文字列を入力してください。: GH3AGa-2435R
入力された文字列は 12 文字です。
文字列を入力してください。: .bye
入力を終了します。
```

課題 5.3.21. 連続する自然数の各位の数の和を考える。例えば、55 から 57 ならば、連続する自然数は $55, 56, 57$ より、$(5+5)+(5+6)+(5+7) = 33$ である。2 つの自然数 a, b $(10 \leqq a < b \leqq 999)$ を入力し、a から b までの各位の数の和を求めるプログラムを作成せよ。

実行結果

```
自然数を入力してください。: 123
123 より大きい自然数を入力してください。: 200
123 から 200 までの各位の数の和は 870 です。
```

課題 5.3.22. (令和 6 年度 四天王寺中学校入試問題 改)

今日は 2024 年 a 月 b 日 c 曜日です。(中略) a, b, c を入力したとき、［ア］～［キ］にあてはまる数 (や曜日) を表示するプログラムを作成せよ。

(1) 2023 年 2 月 22 日は今日から［ア］日前で、［イ］曜日です。

(2) 今日から 500 日後は、［ウ］年［エ］月［オ］日［カ］曜日です。

(3) 2024 年以降で、2 月 29 日が c 曜日になるのは、［キ］年です。

ただし、曜日は日 $= 0$, 月 $= 1$, ..., 土 $= 6$ のように数として扱っても良い。

実行結果

```
年を入力してください。: 2024
月を入力してください。: 6
日を入力してください。: 24
曜日を入力してください。(日=0, 月=1, ..., 土=6): 1
(1) ア=, イ=
(2) ウ=, エ=, オ=, カ=
(3) キ=
```

第 6 章

ふろく

6.1　実習のための覚書

6.1.1　サクラエディタの使い方

応用数学科にインストールされているエディタは、サクラエディタ、Cpad であり、コンパイラとして MinGW または Borland C++Compiler の GCC を使用している。

ちなみに、現在 Borland C++Compiler は一般公開されていない。

(1) デスクトップ上にある サクラエディタのアイコン (右図) をダブルクリックし、起動させる。

(2) 左下の画面 (図 1) が表示されたら、まずファイルを保存 する。

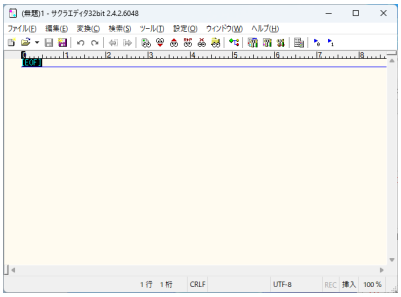


図 1

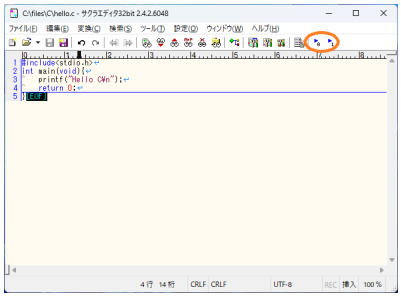


図 2

[ファイル (F)] の [名前を付けて保存 (A)] を選択する。このとき、ファイル名に注意する。

また、ソースファイルの保存場所はD ドライブか、USB メモリに保存すること。

表現とメディアの数理を受講した人は、D ドライブに `¥files` というフォルダがある筈なので、その直下に `C` というフォルダを作成し `D:¥files¥C` に保存するのが良い。

(3) ファイル保存を行った後、入力を開始する。入力後、上書き保存を行う ([Ctrl]+[s])。

(4) 保存が出来たらコンパイル + を行う。図 2 の〇で囲ったアイコン 2 つのうち、左側を押すか、ツールバーの [ツール] から [登録済みマクロ (B)] の [gcc] を選択する。

(5) コンパイルが上手く行けば、図 3 のように表示され、

図 3

エラーが有れば図 4, 図 5 のように表示される。エラーメッセージからエラーを見つけ、修正後に再度コンパイルを行う。

図 4 のエラーは、printf 文の最後に ; が無いというエラーである。図 5 のエラーは、`stdio.h` と書くべき所を `studio.h` と書いている。

図 4

図 5

エラーが出た場合、何かおかしいかメッセージがあるので、それを元に修正して再度、コンパイル + 実行をしてみる。

6.1.2　記号

この講義で使われている記号について、読み方を紹介しておく。

記号	読み方	記号	読み方
|	パイプライン、縦棒	*	アスタリスク
^	ハット	{}	波カッコ, 中カッコ
/	スラッシュ	\	バックスラッシュ
〃	ダブルクォーテーション	.	ドット、ピリオド
′	シングルクォーテーション	,	カンマ, コンマ
:	コロン	;	セミコロン
_	アンダーバー, アンダースコア	@	アットマーク
・	ナカグロ, 中点	!	エクスクラメーションマーク
～	チルダ	#	シャープ, いげた
¥	エンマーク	$	ドルマーク

6.1.3 文字コード

コンピュータで文字を扱う場合、文字そのものを扱うのではなく、文字を表す”文字コード”を用いる。例えば、1Byte で表すことが出来るものとして、ASCII コードと呼ばれる文字コード (0 から 127 まで) があり、A が 65, B が 66, ..., Z が 90 で、a が 97, b が 98, ... となっている。

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
10	LF*	VT	FF*	CR	SO	SI	DLE	DC1	DC2	DC3
20	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
30	RS	US	SP	!	"	#	$	%	&	’
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	‘	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	DEL		

6.1.4 バックスラッシュと円マークについて

文字コードの 92 にある\ (バックスラッシュ) は日本では ¥ が用いられるが、文字コードでは\が表示される。逆に、メーラやエディタでは、¥ が\と表示される場合がある。

Windows PC の日本語キーボードでは右 [Shift] キー周辺の [ろ] のキーに\があるが、押すと¥となる。

♡ **注意!** MacOS を使用している場合は注意が必要で、普通に入力した¥は正しく動作しない ([¥] は¥として機能しない) ことがある。

MacOS では [option] キーを押しながら [¥] キーを押して表示される\を使う。

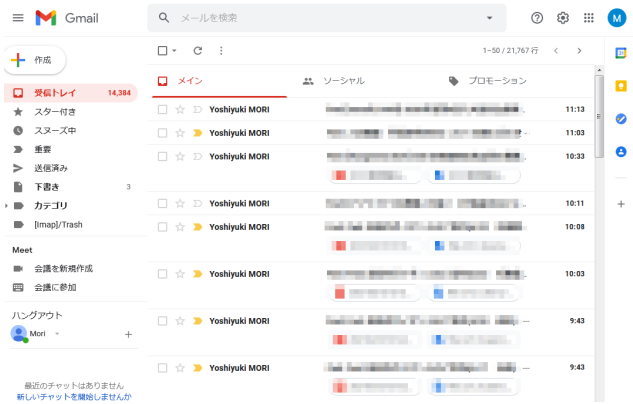
なお、全ての MacOS と、エディタが上記の仕様となっているかは不明なので、最初に使う時に確認してみてもらいたい。

6.2　パソコンでのメールの送信方法

課題提出のためにメールの送信をする必要がある。そのために、OUS-id を用いて Gmail から送信する方法を紹介しておく。

まず、応用数学科のホームページを開くと [在学生の方へ] というリンクがある。このリンクを辿り、下部にある リンク集 から [情報処理センター OUS メール] を選択する。

情報処理センターのページ内の OUS メール [Gmail] を選択すると、Gmail のログイン画面に移る。メールアドレスとして、OUS-id を入力し、パスワードを入力する。正しく入力されると



のような画面が表示される。

メールを作成する場合は左上の [+ 作成] をクリックする。

開いた右図の画面において、計算機とアルゴリズムの課題を提出する場合は

宛先 `mori@xmath.ous.ac.jp`

件名 `S24M000 課題 1.3.5`

の形式とする。場合によっては件名が

件名 `S24M000 課題 2.1.1 課題 2.1.2`

となる。ファイルを添付する場合は、下の欄の  をクリックして、添付するファイルを選ぶ。

6.3 アプリのインストール

はじめに

それぞれのアプリのインストールに関する不具合等は自己責任となるため不安な場合は計算機室のパソコンを使用すること。

6.3.1 C 言語とエディタ

C 言語の実習環境を作るためには、C 言語本体と、エディタをインストールすることが一般的である。エディタと C 言語本体がまとめられたアプリケーション

例えば

Microsoft Visual Studio Community
`https://visualstudio.microsoft.com/ja/vs/`
といものがあるが、” プロジェクト” という概念に困惑するかもしれない。

も存在するが、ここでは C 言語本体とエディタが個別の場合を紹介する。

コンパイラ

まず、C 言語本体 (コンパイラ) は

- Windows の場合、 Mingw-w64
- MacOS の場合、xcode
- Linux の場合 gcc

がある。

MacOS の場合は “Mac での C 言語の環境構築”

`https://hacknote.jp/archives/53322/`

Linux の場合 (Ubuntu の例) は “Ubuntu20.04 gcc をインスールして実行する手順”

`https://mebee.info/2020/06/11/post-12304/`

などを参考にインストールを行う。

エディタ

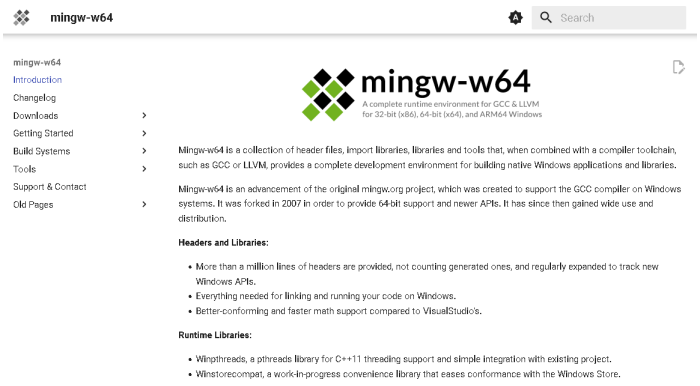
また エディタとして、Windows の場合は Visual Studio Code と サクラエディタ を紹介するので、どちらか一方をインストールする。

MacOS の場合は Visual Studio Code を紹介する。

Linux の場合は各自好みのエディタ (例えば Emacs や vi) がある筈なので、それを使用する。

6.3.2　Mingw-w64 のインストール [MS Windows]

まず、ブラウザで `https://www.mingw-w64.org/` を開く。



このサイトの左側にある [Downloads] のプルから、[Pre-built Toolchains] を選び、移ったサイトを少しスクロールするとある [MingW-W64-builds] をクリックする (図 1)。

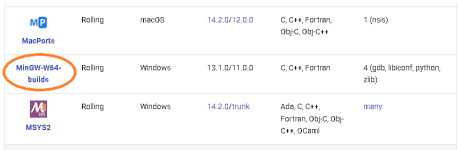


図 1

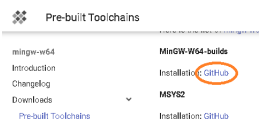


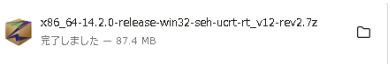
図 2

表示されたサイトの [Mingw-builds] の下の [GitHub] をクリックする (図 2)。

さらに、右のようなサイトに移るので、32bitOS なら、(1) または (2)、64bitOS なら (3) または (4) となり、OS が windows 10 または 11 なら、(2) または (4)、それ以外なら (1) または (3) を選ぶ。

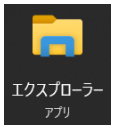
セキュリティソフトを入れている場合、ダウンロードして良いか問合せがあったり、ブラウザによっては保存する場所の問合せがある場合がある。

ダウンロードが終わると



のように表示される (FireFox の場合)。

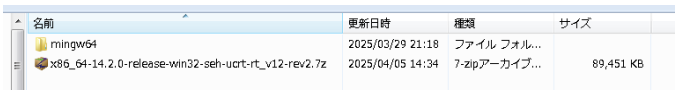
ダウンロードが完了したら ファイル (`*-14.2.0-release-win32-seh-*-rt_v12-rev2.7z`) を解凍する。



ダウンロードしたファイルの保存場所が解らない場合は、エクスプローラーで上記のファイルを探し、見つけて解凍する。ダウンロードされたファイルは、通常 ”ダウンロード” フォルダに保存される。

解凍は最新の Windows 11 ならエクスプローラーから解凍出来るが、7z(7-zip) の解凍に対応していない場合は、新たにアプリをインストールする必要がある。

解凍が終わると以下のように、フォルダ [mingw64] (または [mingw32]) が作成される。

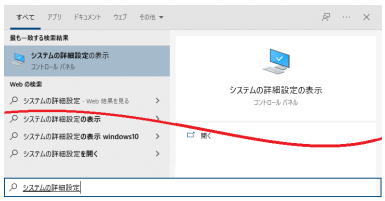


このフォルダを適当な場所に移動させる (※ I)。(計算機室のパソコンでは、`c:¥pf` の元)

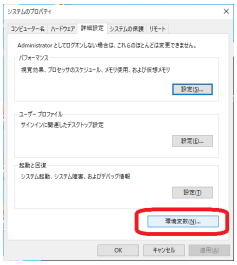
以上で、MinGW のインストールは完了なので、次は環境設定を行う。

環境設定

Windows 11 のタスクバーの検索ボックスに”システムの詳細設定の表示”と入力し、表示されたシステムの詳細設定の表示を選ぶ (画面 3)。

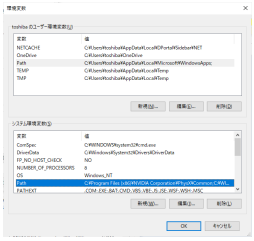


画面 3

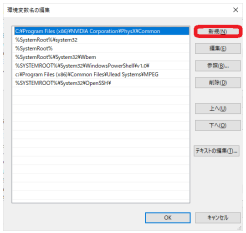


画面 4

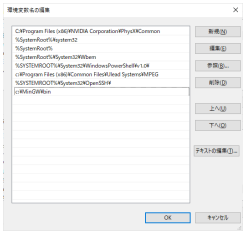
システムのプロパティが表示されたら [詳細設定] から、[環境変数 (N)] を選ぶ (画面 4)。



画面 5



画面 6



画面 7

環境変数のシステム環境変数 (S) から Path を選び、[編集] をクリックする (画面 5)。

環境変数名の変数から [新規 (N)] をクリックし (画面 6)、新たに `c:¥pf¥mingw64¥bin` を追加する (画面 7)。

ただし、(※ I) で他の場所に移動している場合は、**移動したフォルダ** `¥mingw64¥bin` とする。

Windows 11 のタスクバーの検索ボックスに **cmd** と入力する。検索結果から、[コマンドプロンプト アプリ] を選択する。起動したコマンドプロンプトで、

```
gcc --version
```

と入力を行う。バージョン情報が表示されれば、インストールは完了している。

6.3.3　Visual Studio Code [MS Windows], [Mac OS]

まず、Visual Studio Code のダウンロードサイト

```
https://azure.microsoft.com/ja-jp/products/visual-studio-code/
```

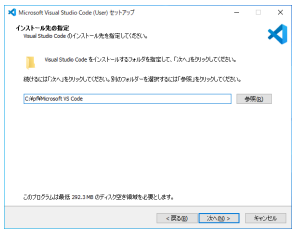
から Visual Studio Code をダウンロードする を選ぶ。

続いて、開いた画面から ↓ Windows または ↓ Mac を選ぶ。
ダウンロード先を聞かれる (場合がある) ので、適当な場所に保存する。
ダウンロードが終わったら、

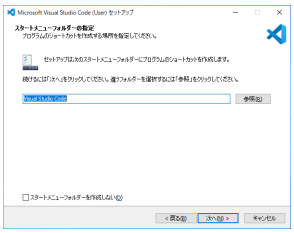
```
VSCodeUserSetup-x64-1.89.0.exe
```

を実行する。

画面 1



画面 2



画面 3

画面 1 が表示されたら [○同意する (A)] にチェックをし、次へ (N)> をクリックする。

まず、インストール先の指定 (画面 2)、スタートメニューフォルダの指定 (画面 3) を指定して 次へ (N)> をクリックする。

次に、追加タスクの選択が表示されるが、□デスクトップ上にアイコンを作成する (D) にチェックし、次へ (N)> をクリックする。

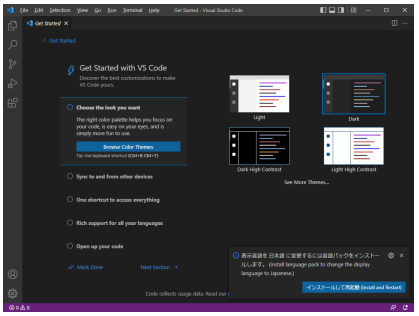
インストール準備完了で、インストール (I) をクリックし、インストールを開始する。

無事インストールが終わると、

[Visual Studio Code セットアップウィザードの完了]

が表示されるので 完了 (F) をクリックする。

次に画面右下 (右図) に表示される指示に従って、言語パック (日本語) をインストールし、Visual Studio Code を再起動する。



再起動後、起動設定を聞かれるので好みで選択する。

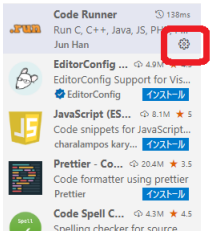
次に、画面左のアイコンの並びから拡張機能 (画面 4 赤枠) を選ぶ。

表示された検索欄に ”code runner” と入力する (画面 4)。

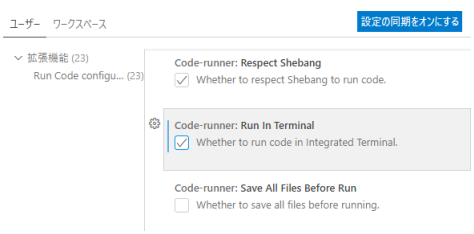
画面 4

”Code Runner”が表示されたら [インストール] をクリックする。

インストールが終了したら、[インストール] があった場所に歯車マークが表示される (画面 5)。このマークをクリックして、設定画面を表示させる。



画面 5



画面 6

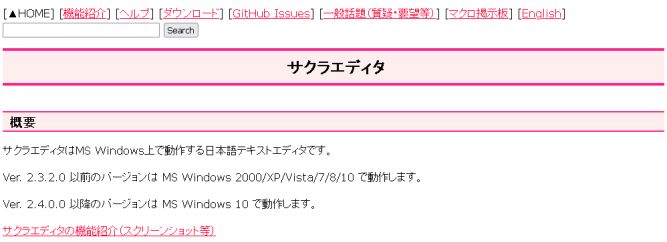
この中から、Code-runner:Run In Terminal を探し、Whether to run code in Integrated Terminal. にチェックをつける (画面 6)。

同様に 

もインストールする。

インストールが終われば、設定等はそのまま閉じてよい。

6.3.4 サクラエディタ [MS Windows]



サクラエディタのサイト (`https://sakura-editor.github.io/`) の [最新版ダウンロード] から、インストーラーをダウンロードする。

2025 年 4 月 10 日現在の最新は、v2.4.2. であり、ファイル名は

`sakura-tag-v2.4.2-build4203-a3e63915b-Win32-Release-Installer.zip`

である。

なお、zip ファイルの展開には圧縮, 解凍ソフトが必要となる (ことがある)。フォルダとして中身が見えないか、インストールされていない場合は、Lhaplus 等のインストールを行う。

Lhaplus で解凍するとデスクトップ上にフォルダが作成され、その中に

`sakura_install2-4-2-6048-x86.exe`

というファイルがある。このファイルを実行すると、インストールが始まる。質問に答えて行けばインストールが終わる。

正しくインストールが終われば、全てのプログラムの中に “サクラエディタ” があるので、起動し、プログラムを作成、保存 (ここでは `hello.c` と) する。

保存後は、コマンドプロンプト (または PowerShell) を起動する。

起動後、ソースファイルを保存した場所に移る (`C:¥files¥C` に保存の場合)。

```
cd ¥files¥C
```

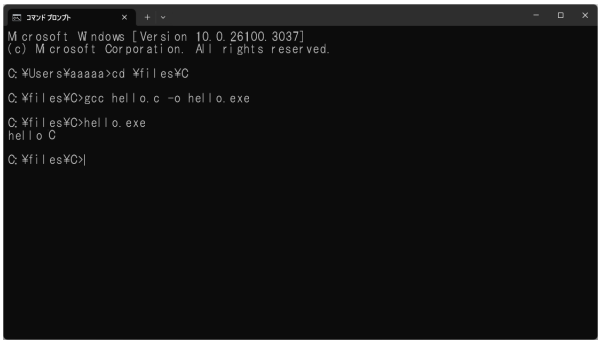
移動後、gcc でコンパイルを行うため

```
gcc hello.c -o hello.exe
```

と入力する。エラーが無ければ、続けて

```
hello.exe
```

と入力すれば実行される。



※ 応用数学科計算機室と同様の仕様にしたい場合は、サクラエディタ, マクロ, カスタマイズ, gcc で調べるか、担当教員に聞く。

6.3.5 セキュリティソフトについて

セキュリティソフトが入っている場合、exe ファイルを消去されることがある。

セキュリティソフトの除外設定が必要になるが、セキュリティソフトによって異なるので、ここではソースネクストの ZERO ウイルスセキュリティの場合で説明する。

また、MinGW のインストール場所と、C 言語のソースファイル保存先の両方で設定が必要な場合もある。

MinGW のインストール場所

1) ウイルスセキュリティのホーム画面を開き、設定をクリックする。
2) ウイルス・スパイウェア対策をクリックする。
3) ウイルス自動検知の中から、”検査したくないファイル、フォルダ”を選択する。
4) 検査除外リストの下にある”ファイルやフォルダを追加”をクリックする。
5) ”+ フォルダを追加する”を選択し、C 言語のソースファイルを置くフォルダを選ぶ。

   例：`C:¥files¥C`

   さらに ”ウイルス自動検知の対象にしない”、”手動検査の対象にしない”を選択し”OK”を押す。

C 言語のソースファイル保存先

6) 表現とメディアの数理 課題 1.1.1 を参考に、C ドライブの直下にサブフォルダ `files` を作成し、 `files` のサブフォルダとして `C` を作成する。
7) 上記の [MinGW のインストール場所] と同様に 1)〜4) を実行し、5) において、

   `C:¥files¥C`

   を選ぶ。また、”ウイルス自動検知の対象にしない”、”手動検査の対象にしない”を選択し”OK”を押す。

ダウンロード先

ユーザーフォルダにあるフォルダ” ダウンロード” に対して上記の設定を行うことは避けるべきである。

そのため、ダウンロードしたファイルが削除される場合は `mingw-get-setup.exe` をダウンロードするためのフォルダを作成し、そのフォルダに対して上記の [MinGW のインストール場所] と同様の作業を行う。

インストール終了後は、そのフォルダを削除しておく。