1 コンピュータプログラミング

1.1 コンピュータプログラミング

プログラム 計算機に実行させる処理手順をある一定の記述言語で具体的に記した表現。

C のような手続き型プログラミング言語では,「所定の目的を達成するために，何をどんな手順で実行する必要があるか（アルゴリズム)」を，記述する必要がある。

プログラミング 計算機のプログラムを作成すること。

1.2 機械語とアセンブリ言語

機械（マシン）語 コンピュータが直接理解できる「ことば」。0 と 1 の並びでできている。コンピュータ (CPU) の種類によって異なる。

1300042774
1400593419
1200274027

アセンブリ言語 プログラミング言語の一種。アセンブリ言語の命令と機械語の命令はほぼ一対一の対応をもつ。

LOAD BASEPAY
ADD OVERPAY
STORE GROSSPAY

アセンブラ アセンブリ言語から機械語への翻訳プログラム

高級言語 (高水準言語) 人間が使う言語に近い要素に基づくプログラミング言語の総称

grossPay = basePay + overTimePay

コンパイラ 高級言語で書かれたプログラムを機械語に一括して翻訳するプログラム[1]

1.3 高水準言語

FORTRAN (1957) 数値計算，技術計算

LISP (1959) 記号処理，人工知能

COBOL (1960) 事務処理

BASIC (1965) 汎用，初心者向け

Pascal (1971) プログラミング教育用

C (1971) システム記述，汎用

1989 年 ANSI (American National Standards Institute; 米国標準規格協会) による C 言語の標準化

[1] これに対し，作成したプログラムを機械語に翻訳しながら逐次実行するプログラムを**インタープリタ**といいます。

Prolog (1973) 論理プログラミング，人工知能，自然言語の構文解析

Ada (1983) 信頼性や保守の容易性に優れる。システムプログラム向き

C++ (1985) オブジェクト指向プログラミングが容易になるように C を拡張

Java (1996) C 言語を基盤とした構文; オブジェクト指向; 汎用

この他にもたくさんのプログラミング言語があります。

2 C プログラムの作成から実行まで

以下は UNIX 系 OS における方法であるが，Windows 等で行う場合も基本的な流れは同じである。

1. エディタを使ってソースコード (source code) を書き，ファイルに保存する。このファイルをソースファイル (source file) という。C のソースファイルでは，ファイル名の終りに `.c` を含める。

   - vi や vim エディタの起動方法例

     `vim` *filename*

     filename は適当なソースファイル名に置き換えること。

   - GNU Emacs の起動方法例 (X Window System が使える場合)

     ```
     emacs &
     ```

     Emacs が起動したら，まず作成するファイルを開いてから編集する。

2. ソースファイルをコンパイル (compile) して実行可能ファイル (executable file) を作る[2]。コンパイルを行うプログラム (コマンド) をコンパイラ (compiler) という。この授業で使うコンピュータには GNU C compiler がインストールされており，起動のためのコマンド名は gcc または cc である。

   `gcc` *filename*

   コンパイルの途中でエラーメッセージが出たら，1. に戻ってソースファイルを修正する。

   エラーが無ければ，a.out という実行可能ファイルができあがる。

3. プログラムを実行し，実行結果を確認する。

   ```
   ./a.out
   ```

   プログラム実行時にエラーが発生したり，実行結果に不具合があれば，1. に戻ってソースファイルを修正し，2. のコンパイルも行う。

   なお，プログラムに潜む欠陥をバグ (bug) といい，バグを取り除く (プログラムを修正する) 作業をデバッグ (debug) という。

[2] 正確には，コンパイルの後にリンク (link) が行われて実行可能ファイルが出来上がるのであるが，ここでの「コンパイル」という言葉には，リンクのステップも含むこととする。

2.1 例題

1. **プログラムのコンパイル・修正・実行**: 次の内容をもつソースファイル ex1.c を作成し，コンパイルしなさい。

```
/* A first program in C */
#include <stdio.h>

int main()
{
    printf("Hello world!\n")

    return 0;
}
```

コンパイル時にエラーメッセージが出ることを確認したら，printf で始まる行の行末に ; (セミコロン) を加えてから再度コンパイルし，プログラムを実行しなさい。画面に Hello world! と表示されるはずです。

2. **プログラム実行時のエラー**: 次の内容のソースファイルを ex2.c として作成しなさい。

```
/* Runtime error */
#include <stdio.h>

int main()
{
     int i = 2;

     printf("6 / %d = %d\n", i, 6 / i);

     return 0;
}
```

その後，下記の指示に従いなさい。

(a) ソースファイルをコンパイルし，エラーメッセージが表示されないことを確認してから，このプログラムを実行しなさい。6 / 2 の計算結果が表示されます。

(b) ソースファイル中で 2 と記述されている箇所を 0 に変更し，同じファイル名で上書き保存しなさい。このソースファイルをコンパイル・実行し，実行時にエラーメッセージが出ることを確認しなさい。

> コンパイルに成功しても，作成したプログラムが正しいとは限りません。プログラムの実行時にエラーが起きた場合にも，ソースファイルを修正してコンパイルし直す必要があります。

(c) ソースファイル中の 0 を 3 に変更してからコンパイルし，正しく実行できることを確認しなさい。

3. 1. のソースコードを参考にして，画面に自分の学生番号と氏名（ローマ字でよい）を出力するプログラムを作成しなさい。ソースファイル名は ex3.c とする。

A　vi (vim) のモードと主要コマンド

i, **a**, **o**, **O**, **cw**, **cc** ...

⟶

コマンド入力モード　　文字入力モード

⟵

<ESC>

機能	コマンド	コメント
テキストを入力する	**i** *text* <ESC>	カーソルの位置に *text* を挿入する (insert)
	a *text* <ESC>	カーソルの右に *text* を追加する (append)
	o *text* <ESC>	今いる（カーソルがある）行の下に *text* を入れる
	O *text* <ESC>	今いる行の上に *text* を入れる
ファイルを保存する	**:w** <RET>	編集中のファイルを元の名前のまま保存する
	:w *file* <RET>	編集中のファイルを *file* として保存する
vi を出る	**ZZ** (**:x** <RET>)	ファイルを保存し vi を出る
	:q! <RET>	ファイルを保存せず vi を出る
カーソルを動かす	**h** (←)	左隣に移動する
	l エル (<SPC> または →)	右隣に移動する
	k (↑)	上に移動する
	j (<RET> または ↓)	下に移動する
	0 ゼロ	行頭に移動する
	$	行末に移動する
	G	最後の行に移動する
	n **G**	第 *n* 行に移動する
行を連結する	**J**	今いる行（カーソルが乗っている文字）に次の行を連結する
テキストを削除する	**x**	今いる文字（カーソルが乗っている文字）を削除する
	X	カーソルの左隣の文字を削除する
	dw	今いる単語を削除する
	dd	今いる行を削除する
テキストを検索する	/ *string* <RET>	*string* が最初に現れる位置にカーソルを移動する
	n	一番最近行った検索を繰り返す
テキストを置換する	**r** *character*	今いる文字を *character* で置換する
	cw *text* <ESC>	今いる単語を *text* で置換する
	cc *text* <ESC>	今いる行を *text* で置換する
変更を繰り返す	**.**	直前のコマンドによる変更を繰り返す
変更を取り消す	**u**	直前のコマンドによる変更を取り消す (undo)
テキストをコピーする	**yy**	今いる行を名前なしバッファにコピー（ヤンク）する
	p	名前なしバッファ中のテキストを挿入（プット）する

括弧 () は当該操作を他のコマンドで行えることを表す。

斜体字 (*italic*) の箇所は具体的なものに置き換えて記述する。例えば，*text* には入力テキストを，*file* にはファイル名を書く。

B GNU Emacs の主要コマンド

C-x C-c	終了
C-g	コマンドの取り消し
C-x u	変更の取り消し (undo)
C-_	変更の取り消し (undo)
M-x help <RET>	ヘルプ
M-x help <RET> t	チュートリアル
C-x C-f	ファイルを開く (バッファへのファイル読込み)
C-x C-s	現在のバッファをファイルに保存
C-x C-w	現在のバッファに名前をつけて保存
C-x s	編集中のバッファをすべてファイルに保存
C-d	カーソル位置の 1 文字を削除
C-k	行末まで消去 (kill)
C-y	最後に保存した kill-ring の内容取りだし (yank)
C-s	検索
M-%	置換
C-a	カーソルを行頭に移動
C-e	カーソルを行末に移動
C-v	次の画面を見る
M-v	前の画面を見る

C UNIX コマンド

分類	コマンド	機能
オンラインマニュアル	man *command*	*command* のマニュアルを表示
ディレクトリ操作	ls	カレント・ディレクトリに存在するファイルの名前を表示
	cd *directory*	カレント・ディレクトリを *directory* に変更
	pwd	カレント・ディレクトリ名の表示
	mkdir *directory*	*directory* の作成
	rmdir *directory*	*directory* の削除
	mv *directory1 directory2*	*directory1* の名前を *directory2* に変更
ファイル操作	cp *file1 file2*	*file1* を *file2* に複写
	cp *files directory*	*files* を *directory* に複写
	mv *files directory*	*files* を *directory* に移動
	mv *file1 file2*	*file1* の名前を *file2* に変更
	rm *files*	*files* を削除
その他	cat *files*	*files* の内容を表示
	more *files*	*files* の内容を一画面毎に表示
	file *files*	*files* の種類を表示
	diff *file1 file2*	*file1* と *file2* の違いを表示