【C言語】入門

基礎文法編

■初歩的注意

- ▶ ※ 大文字と小文字を区別する言語である。

■データ型

- 整数　int　long　long long　※前に `unsigned` をつければ0以上の数だけ
- 実数　float　double
- 文字　char
- 値なし　void　※特別な型

■基礎

- ▶ コメントのしかた
- ▶ ☆ テンプレ（メイン関数）
- ▶ 変数を定義
- ▶ 静的変数を定義
- ▶ レジスタ変数を定義
- ▶ 関数内の定数を定義
- ▶ 整数値の定数を定義
- ▶ 実行中のCファの名前
- ▶ 実行中の関数の名前

■プリプロセッサ

- ▶ ※ C言語では、コンパイルの前に cpp というプリプロセッサによる前処理を行う。なお、文の最後に `;` は不要であることに注意。
- ▶ ヘッダファイル読込み
- ▶ 自作の〃読込み
- ▶ 定数を定義
- ▶ マクロを定義
- ▶ 定数やマクロを削除

■標準入出力

【C言語】入門

基礎文法編

■初歩的注意

- ▶ ※ 大文字と小文字を区別する言語である。

■データ型

- 整数　int　long　long long　※前に `unsigned` をつければ0以上の数だけ
- 実数　float　double
- 文字　char
- 値なし　void　※特別な型

■基礎

- ▶ コメントのしかた　/* */ で囲めば改行可能
- ▶ ☆ テンプレ（メイン関数）
- ▶ 変数を定義　型名 a = 値;　型名 a;　型名 a=値, b=値;　型名 a, b;　のどれか
- ▶ 静的変数を定義　関数内で　static 型名 a = 値;　や　static 型名 a;
- ▶ レジスタ変数を定義　register 型名 a = 値;　register 型名 a;
- ▶ 関数内の定数を定義　const HOGE = 値;　※一応ソースファ冒頭に書いても使える
- ▶ 整数値の定数を定義　enum {A, B, C,}　や　enum {A, B = 4, C,}　※0,1,2 や 0,4,5となる
- ▶ 実行中のCファの名前　__FILE__　※ `__LINE__` でその行数を知れる
- ▶ 実行中の関数の名前　__func__

■プリプロセッサ

- ▶ ※ C言語では、コンパイルの前に cpp というプリプロセッサによる前処理を行う。なお、文の最後に `;` は不要であることに注意。
- ▶ ヘッダファイル読込み　#include <*header*.h>
- ▶ 自作の〃読込み　#include "*header*.h"　※真っ先にカレディを探すようになる
- ▶ 定数を定義　#define HOGE 値
- ▶ マクロを定義　#define HOGE(p1, p2) コード様の文字の羅列
- ▶ 定数やマクロを削除　#undef HOGE

■標準入出力

- ▶ １文字を入力
- ▶ 文字列を入力

- ▶ １文字を出力
- ▶ 文字列を出力
- ▶ 変数展開して出力

■条件分岐

- ▶ 条件分岐
- ▶ 条件演算子
- ▶ 論理演算子
- ▶ ２股分岐の略記
- ▶ switch文

■繰り返し処理

- ▶ *n*回処理を繰り返す
- ▶ while文
- ▶ do-while文
- ▶ 中断し、次へ・脱出

■その他の制御

- ▶ ラベルへジャンプ

■関数

- ▶ 関数を定義
- ▶ 仮引数がない場合
- ▶ 返り値がない場合

- ▶ 関数の呼び出し

- ▶ ☆ スコープの話――外部変数、自動変数、静的変数
- ▶ ☆ 引数を参照渡しする

■例外処理

- ▶ 強制終了

■数値

- ▶ １文字を入力　**int** c;　c = **getchar**();　※: 文字の数値; `EOF` (= `-1`)
- ▶ 文字列を入力　char str[十分な字数];　**gets**(str);　※: なし

- ▶ １文字を出力　**int** c;　c = 'a';　**putchar**(c);　※: 文字の数値; `EOF` (= `-1`)
- ▶ 文字列を出力　char str[] = "abc";　**puts**(str);　※: なし
- ▶ 変数展開して出力　printf("書式文字列", 変数1, 変数2, ...);　※: 文字数; `EOF` (= `-1`)

■条件分岐

- ▶ 条件分岐　if　**else if**　else
- ▶ 条件演算子　<　<=　>　>=　==　!=
- ▶ 論理演算子　&&　||　!　!( )
- ▶ ２股分岐の略記　条件式 ? 真での値 : 偽での値
- ▶ switch文　switch (何か) { case 値: 処理; **break;**　...　default: 処理; }

■繰り返し処理

- ▶ *n*回処理を繰り返す　int i;　for (i = 1; i <= *n*; i++) { 処理 }
- ▶ while文　while (条件式) { 処理;　**条件に関する処理**; }
- ▶ do-while文　do { 処理;　**条件の処理**; } while (条件式);　※一度は必ず実行
- ▶ 中断し、次へ・脱出　continue;　・　break;

■その他の制御

- ▶ ラベルへジャンプ　goto *label*;　※ `Label:` でどこかにラベルを設けておく

■関数

- ▶ 関数を定義　返り値の型 関数名(引数の型1 仮引数1, ...);　※この宣言が必須
  返り値の型 関数名(引数の型1 仮引数1, ...){ ‥　return 返り値; }
- ▶ 仮引数がない場合　返り値の型 関数名(void);　返り値の型 関数名(void){ 〃 }
- ▶ 返り値がない場合　void 関数名(‥);　void 関数名(‥){ ‥ };

- ▶ 関数の呼び出し　関数(実引数1, ‥)　※ 末尾に `;` が必要なことも当然ある

- ▶ ☆ スコープの話――外部変数、自動変数、静的変数
- ▶ ☆ 引数を参照渡しする

■例外処理

- ▶ 強制終了　exit(0);

■数値

- ▶ 算術演算子
- ▶ 複合代入演算子
- ▶ ｲﾝｸﾘﾒﾝﾄ,ﾃﾞｸﾘﾒﾝﾄ演算子
- ▶ m×10^n で表現

■配列

- ▶ ※ 配列は、**同じ型**の変数の集まり（順番あり）である。
- ▶ 配列を定義
- ▶ 要素の値を取得
- ▶ 要素の値を変更
- ▶ 要素数
- ▶ 配列をコピー
- ▶ 多次元配列

■文字・文字列

- ▶ ※ 文字は `''` で、文字列は `""` でくくる。
- ▶ 特殊な文字を表現
- ▶ 文字を変数に格納
- ▶ 文字列を(配列)変数に代入
- ▶ 変数展開

■構造体

- ▶ ※ 構造体は、**型が異なるものも含めて**、複数の変数を１パッケージ（順番あり）として扱うものである。
- ▶ 構造体テンプレを宣言
- ▶ 構造体を変数に格納

■ファイル操作

- ▶ 開く
- ▶ １文字読み取る
- ▶ １行読み取る

- ▶ 算術演算子　+　-　*　/　%
- ▶ 複合代入演算子　+=　-=　*=　/=　%=
- ▶ ｲﾝｸﾘﾒﾝﾄ,ﾃﾞｸﾘﾒﾝﾄ演算子　a++　++a　a--　--a
- ▶ m×10^n で表現　1.23e-4

■配列

- ▶ ※ 配列は、**同じ型**の変数の集まり（順番あり）である。
- ▶ 配列を定義　型名 配列名[] = {値1, 値2, 値3, ...};　か　型名 配列名[要素数];
- ▶ 要素の値を取得　配列[番号]
- ▶ 要素の値を変更　配列[番号] = 値;
- ▶ 要素数　sizeof(配列) / sizeof(配列[0])
- ▶ 配列をコピー　#include <string.h>　a2を定義;　memcpy(a2, a1, sizeof(a1));
- ▶ 多次元配列　型名 配列[行数][列数];　などで定義
配列[行番号][列番号]...　で要素の値にアクセス

■文字・文字列

- ▶ ※ 文字は `''` で、文字列は `""` でくくる。
- ▶ 特殊な文字を表現　\n　\t
- ▶ 文字を変数に格納　char c = 'A';　※ 全角文字は入れられない。
- ▶ 文字列を(配列)変数に代入　char 配列名[] = "Hello";　か　char 配列名[最大字数] = "Hello";
か　char 配列名[] = {'H', 'e', 'l', 'l', 'o', **'\0'**};
- ▶ 変数展開　sprintf(代入先の変数, "書式文字列", 変数1, 変数2, ...);

■構造体

- ▶ ※ 構造体は、**型が異なるものも含めて**、複数の変数を１パッケージ（順番あり）として扱うものである。
- ▶ 構造体テンプレを宣言　struct 構造体名 {　ﾒﾝﾊﾞの型名1 ﾒﾝﾊﾞ名1;
ﾒﾝﾊﾞの型名2 ﾒﾝﾊﾞ名2[要素数];　...　};
- ▶ 構造体を変数に格納　struct 構造体※ 変数名;　※さっき宣言した構造体テンプレ名
変数.ﾒﾝﾊﾞ1 = 値1;　変数.ﾒﾝﾊﾞ2 = { 値2_1, 値2_2, ... };　...

■ファイル操作

- ▶ 開く　FILE *fp;　fp = **fopen**("～.txt", "*mode*");　※: ストリーム; `NULL`
- ▶ １文字読み取る　**int** c;　c = **getc**(fp);　※: 文字の数値; `EOF` (= `-1`)
- ▶ １行読み取る　**fgets**(代入先の配列, 最大字数, fp);　※: 配列; `NULL`

- ▶ １文字上書きか追記
- ▶ 文字列を上書きか追記
- ▶ 閉じる

■ポインタ

- ▶ ポインタ変数とは
- ▶ ※ ポインタがもつデータは、先頭アドレスと、記憶領域の大きさ。
- ▶ ポインタ変数を宣言
- ▶ 変数のアドを取得
- ▶ アドが指す変数の値

- ▶ １文字上書きか追記　**int** c;　c = 'a';　**putc**(c, fp);　※: 非負数; `EOF` (= `-1`)
- ▶ 文字列を上書きか追記　char str[] = "abc";　**fputs**(str, fp);　※: 非負数; `EOF` (= `-1`)
- ▶ 閉じる　**fclose**(fp);　※: `0`; `EOF` (= `-1`)

■ポインタ

- ▶ ポインタ変数とは　値としてアドレスをもつ変数
- ▶ ※ ポインタがもつデータは、先頭アドレスと、記憶領域の大きさ。
- ▶ ポインタ変数を宣言　型※ *変数名;　※どのような型のデータへのポインタであるか
- ▶ 変数のアドを取得　&変数
- ▶ アドが指す変数の値　*アド