飛翔ソフトウェア

新人研修

C言語基礎編

|  |  |  |
| --- | --- | --- |
| 日付 | 変更内容 | 変更者 |
| 2021/02/03 | 新規作成 | 庭 |
|  |  |  |

# 研修の目的

飛翔ソフトウェアが受注している仕事にはC言語での開発が多く、必須スキルとなります。また、C言語の基礎を学ぶことで他言語の習得もスムーズになります。

# はじめに

まずは以下のサイトや社内の参考書を読み、C言語の学習をしましょう。

苦しんで覚えるＣ言語

<https://9cguide.appspot.com/index.html>

# コーディングルール

本研修でのコーディングルールを規定します。

職場毎にコーディングルールは変わるため、研修後は配置された職場のコーディングルールに従いましょう。

## インデント

インデントとは、ブロックの始まりと終わりをはっきりさせるためのものです。

インデントにはタブを使用し、タブは4文字分とします。

半角スペースを使用する方法や、4文字ではなく8文字など、しばしば論争になりますが、本研修ではタブで4文字分とします。

プログラムが右に行き過ぎて読みにくくなってしまう場合、プログラムそのものが良くない可能性があります。ネストが深くならないよう、修正を検討してください。

switch 文は、「switch」とそれに含まれる「case」ラベルを同じ列にそろえます。

|  |
| --- |
| switch (suffix) {  case 'G':  case 'g':  mem <<= 30;  break;  case 'M':  case 'm':  mem <<= 20;  break;  case 'K':  case 'k':  mem <<= 10;  /\* fall through \*/  default:  break;  } |

何か隠したいことがない限り、１行に複数の文を書かないでください。

|  |
| --- |
| if (condition) do\_this;  do\_something\_everytime; |

## 括弧の位置と空白

インデントの場合とは違い、どこに括弧を置くのが優れているかというような技術的な根拠はほとんどありません。それでも好ましい括弧の位置というのはあって、ブロックの開始行の行末に開始括弧を置いて、ブロックの最終行の始まりに終了括弧を置くというもので、次のようになります。

これは、関数以外のブロック全て（if、switch、for、while、do）にあてはまります。

|  |
| --- |
| if (x is true) {  we do y  } |

ただし、関数定義の括弧だけは例外で、開始括弧は次の行の始まりに置きます。

|  |
| --- |
| int function(int x)  {  body of function  } |

ここで一つ注意しておきます。終了括弧は普通、終了括弧だけの行になりますが、終了括弧で文が終わらない場合にはその行が続くことになります。すなわち do 文では "while" が、 if 文では "else" が来ることがあります。

|  |
| --- |
| do {  body of do-loop  } while (condition);  if (x == y) {  ..  } else if (x > y) {  ...  } else {  ....  } |

以下のキーワードの後には、空白を使ってください。

if, switch, case, for, do, while

しかし、sizeof、typeof、alignof、\_\_attribute\_\_

の後には空白を使わないでください。

括弧内の式の前後に、空白を使わないでください。

|  |
| --- |
| s = sizeof( struct file );  s = sizeof(struct file); |

ポインタデータ、またはポインタ型を返す関数を定義するとき、

「\*」はデータ名または関数名にはつけて、型名につけません。

|  |
| --- |
| char \*linux\_banner;  unsigned long long memparse(char \*ptr, char \*\*retptr);  char \*match\_strdup(substring\_t \*s); |

以下のように、たいていの二項演算子や三項演算子の演算子の前後に、空白を１文字使ってください。

= + - < > \* / % | & ^ <= >= == != ? :

|  |
| --- |
| c=a+b;  if (c<=MAX) {  c=0;  }  c = a + b;  if (c <= MAX) {  c = 0;  } |

しかし単項演算子の後には、空白を使わないでください。

& \* + - ~ ! sizeof typeof alignof \_\_attribute\_\_ defined

|  |
| --- |
| a = sizeof (long);  a = - a;  a = sizeof(long);  a = -a; |

後置インクリメント単項演算子や後置デクリメント単項演算子の前には、空白を使わないでください。

++ --

前置インクリメント単項演算子や前置デクリメント単項演算子の後に空白を使わないでください。

++ --

|  |
| --- |
| c = 0;  c ++;  ++ c;  c = 0;  c++;  ++c; |

構造体演算子「.」と「->」の前後に、空白を使わないでください。

行末に無駄な空白を残さないでください。

## 名前の選択

グローバル変数（本当に必要な時にだけ使うこと）には意味がよく分かる名前を付けてください。グローバル関数も同じです。

ローカル変数の名前は短くて、変数の特徴を表したものにします。

## 関数

関数は短くて親切、かつ単純に一つのことを行いましょう。

一つの関数は一つの処理に専念して、それだけをキッチリと行いましょう。

関数内のローカル変数の数も目安となります。ローカル変数は５〜１０個にとどめるべきもので、それを超えているようなら何か悪いことをしているのです。そういう場合は関数を見直し、より細かく分割してください。

ソースファイル内では、空行１行で関数を区切ってください。

関数のプロトタイプ宣言には、データの型と一緒に、パラメータ名を含めてください。

これはＣ言語にとって、仕様上、必要ではありませんが、読み手にとって価値のある情報を加える簡単な方法なので、推奨されます。

## 関数終了コードの共通化

goto 構文は、関数がいくつかの場所で処理を終了してから共通的な動作（例えばクリーンアップ動作）を行う場合に使用します。それ以外で安易に使用しないでください。

|  |
| --- |
| int fun(int a)  {  int result = 0;  char \*buffer = malloc(SIZE);  if (buffer == NULL)  return -ENOMEM;  if (condition1) {  while (loop1) {  ...  }  result = 1;  goto out;  }  ...  out:  kfree(buffer);  return result;  } |

## コメント

コメントスタイルは、C89 の "/\* ... \*/" スタイルです。

C99 スタイルの "// ..." のコメントは使用してはいけません。

長い（複数行にわたる）コメントにとって推奨されるスタイルを次に示します。

|  |
| --- |
| /\*  \* この形式が、複数行のコメントにおいて  \* 推奨されるスタイルです。  \* 一貫して、それを使用してください。  \*  \* 説明 - 左側にアスタリスクの列を置き、開始行と終了行には、  \* なにもない行を置いてください。  \*/ |

## マクロ、列挙型

定数を定義するマクロの名前と列挙定数は大文字で記述します。

|  |
| --- |
| #define CONSTANT 0x12345 |

列挙型は複数の関連する定数を定義するときに好んで利用されます。

マクロは大文字が好ましいですが、関数形式マクロは小文字でも構いません。

一般的に、関数形式マクロよりもインライン関数の方が望ましいでしょう。

複数の文から構成されるマクロは do - while ブロックで囲むべきです。

|  |
| --- |
| #define macrofun(a, b, c) \  do { \  if (a == 5) \  do\_this(b, c); \  } while (0) |

マクロ使用時に避けるべきこと

1. 制御の流れに影響を与えるマクロ

|  |
| --- |
| #define FOO(x) \  do { \  if (blah(x) < 0) \  return -EBUGGERED; \  } while(0) |

関数呼び出しのように見えながら、「呼んだ」関数から抜けてしまいます。

1. 特定の名前のローカル変数に依存しているマクロ

|  |
| --- |
| #define FOO(val) bar(index, val) |

コードを読む際に混乱させるし、表面上、問題の無い変更が処理を壊す傾向にあります。

1. 左辺値として使われる引数付マクロ

|  |
| --- |
| FOO(x) = y |

1. 演算子の優先度について忘れること

式で定数を定義するマクロは式を括弧で囲まなければなりません。引数を利用するマクロの同じような問題に用心しましょう。

|  |
| --- |
| #define CONSTANT 0x4000  #define CONSTEXP (CONSTANT | 3) |

# コーディング作法

IPA 独立行政法人 情報処理推進機構が発行しているコーディング作法ガイドが参考になりますので参考にしてください。

<https://www.ipa.go.jp/files/000064005.pdf>

## インデントが深くなってしまったら

|  |
| --- |
| int example\_function(void)  {  int result = 0;  int error = 0;    result = function1();  if (result < 0) {  error = -EACCES;  } else {  result = function2();  if (result < 0) {  error = -EBUSY;  } else {  result = function3();  if (result < 0) {  error = -ENOMEM;  }  }  }  return error;  }  int example\_function(void)  {  int result = 0;  int error = 0;    do {  do while文で囲み、条件が成立したらbreakすることでネストを浅くできます。  result = function1();  if (result < 0) {  error = -EACCES;  break;  }  result = function2();  if (result < 0) {  error = -EBUSY;  break;  }  result = function3();  if (result < 0) {  error = -ENOMEM;  break;  }  } while (0);  return error;  } |

## 途中return

気を付けて使用しましょう。

|  |
| --- |
| int example\_function(void)  関数の途中でreturnすると実行すべき処理が抜けてしまう可能性があります。バグを引き起こしやすいため避けましょう。  {  int result = 0;  char \*buff = NULL;  buff = malloc();  if (buff == NULL) {  return -ENOMEM;  }  result = function2(buff);  if (result < 0) {  return -EBUSY; **/\* ここでreturnするとfreeが呼ばれなくなってしまう \*/**  }  free(buff);  return 0;  }  int example\_function(void)  {  int result = 0;  int error = 0;  char \*buff = NULL;  do {  buff = malloc();  if (buff == NULL) {  error = -ENOMEM;  break;  }  result = function2(buff);  if (result < 0) {  error = -EBUSY;  break;  }  } while (0);  if (buff != NULL) {  free(buff);  }  return 0;  } |

|  |
| --- |
| int example\_function(int type)  {  switch (type) {  switch文の中にreturnを入れるのも避けましょう。  case TYPE\_A:  return 1;  break;  case TYPE\_B:  return 2;  break;  default:  return 3;  break;  }  return 0;  }  int example\_function(void)  {  int result = 0;  int error = 0;  char \*buff = NULL;  do {  もちろんこのようなreturnもダメです。  buff = malloc();  if (buff == NULL) {  return -ENOMEM;  break;  }  result = function2(buff);  if (result < 0) {  return -EBUSY;  break;  }  } while (0);  if (buff != NULL) {  free(buff);  }  return 0;  } |

|  |
| --- |
| int example\_function(int type)  {  if (type >= TYPE\_MAX) {  return -1;  処理前のパラメータチェックでのエラー時に使用すると、後述の処理がスッキリしたりするのでよいです。  }  switch (type) {  case TYPE\_A:  break;  case TYPE\_B:  break;  default:  break;  }  return 0;  } |

## マジックナンバーを避ける

wikipediaより引用

|  |
| --- |
| プログラムにおけるマジックナンバー（英: magic number、魔法の数字）とは、何らかの識別子もしくは定数として用いられる、プログラムのソースコード中に書かれた具体的な数値である。そのプログラムを書いた時点では製作者は数値の意図を把握しているが、他のプログラマーまたは製作者本人がマジックナンバーの意図を忘れたときに閲覧すると「この数字の意味はわからないが、とにかくプログラムは正しく動く。まるで魔法の数字だ」という皮肉を含む。 |

|  |
| --- |
| int example\_function(int type)  {  int result = 0;  switch(type) {  case 0:  break;  case 1:  break;  default:  break;  }  return result;  }  typedef enum {  EXAMPLE\_TYPE\_A = 0,  EXAMPLE\_TYPE\_B,  enumやdefineを使って読み手に分かりやすくします。  また、値の変更が容易になり、保守性も上がります。  EXAMPLE\_TYPE\_MAX,  } EXAMPLE\_TYPE;  int example\_function(EXAMPLE\_TYPE type)  {  int result = 0;  switch(type) {  case EXAMPLE\_TYPE\_A:  break;  case EXAMPLE\_TYPE\_B:  break;  default:  break;  }  return result;  } |

## 型の選択

変数を作成する際、何の型にすればよいだろう？

と迷うことがあるかもしれません。

一例をあげておきます。

|  |  |
| --- | --- |
| int | 最も使われる型かもしれません。例えば、関数の戻り値として使用し、成功なら0/失敗なら不値として使用することが多いです。その他にもさまざまな変数に使用されます。 |
| unsigned int | intと違って不値がありませんが、こちらもさまざまな変数に使用されます。 |
| char | 文字列を扱う場合に使用されます。 |
| short | 2byteのデータを扱う場合に使用されます。それ以外にはあまり使われません。 |
| long | intの代わりによく使われますが、32bitCPUでは4byte、64bitCPUでは8byteにサイズが変わります。多用すると痛い目にあいます。 |
| long long | 8byteのサイズを使用したいのであれば、longを使用せずにlong longを使用しましょう。 |
| float/double | 複雑な算術を使用する場合以外には使いません。今回の研修で使用することはないでしょう。 |
| bool | true/falseの二択なので、なにかを判定する関数の戻り値としてよく使用されます。 |

## フォールスルー

switch文などのbreakをあえて入れない場合は、それがわかるように/\* fall through \*/とコメントをいれて、break抜けのバグなのか意図的なのかがわかるようにしましょう。

|  |
| --- |
| switch (suffix) {  case 'G':  case 'g':  mem <<= 30;  break;  case 'M':  case 'm':  mem <<= 20;  break;  case 'K':  case 'k':  mem <<= 10;  /\* fall through \*/  default:  break;  } |

## 多重インクルード防止

ヘッダファイルには「自分はすでにインクルードされています」を意味するシンボルを定義(#define)して、ヘッダ全体を次のようにインクルードガードで囲みましょう。

下記例のHEADER\_Hは各自のヘッダファイル名に置き換えてください。

|  |
| --- |
| #ifndef HEADER\_H  #define HEADER\_H  /\* ... <header.h> の内容がここに入る \*/  #endif /\* HEADER\_H \*/ |

これがないと、ヘッダファイル内の定義が多重定義されてしまう可能性があります。

## TBD

適宜追加予定

# 実習

実際にC言語を使用してプログラムを作成しましょう。

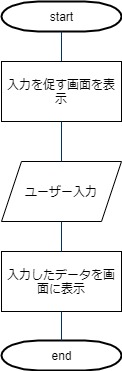
開発環境にはVisual Studioを使用します。

## 課題1

以下のフローチャートとプログラムを改造してください。

改造内容：ユーザー入力関数の戻り値も画面に出力してください。

フローチャート



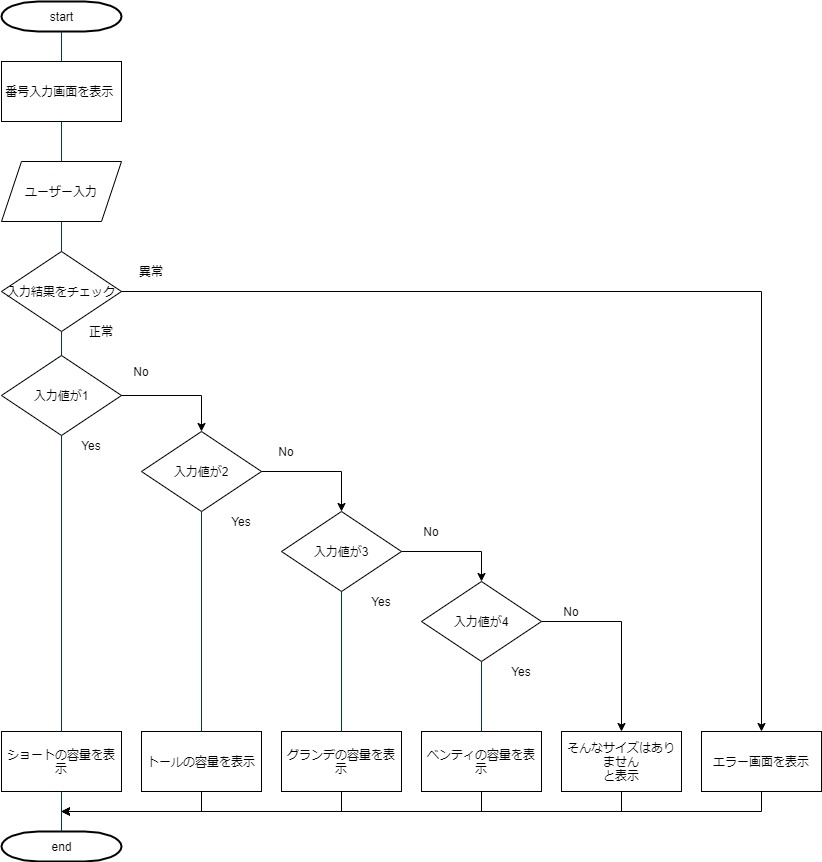
プログラム

|  |
| --- |
| #include <stdio.h>  #define TEST\_BUFF\_SIZE (50 + 1)  void main(void)  {  char buff[TEST\_BUFF\_SIZE] = {"\0"};  /\* 入力を促す画面を表示 \*/  printf("scanf\_sの動作確認です。文字を入力してください。\n");  printf(":");  /\* ユーザー入力 \*/  scanf\_s("%s", buff, sizeof(buff));  /\* 結果を表示 \*/  printf("出力バッファ:%s", buff);  } |

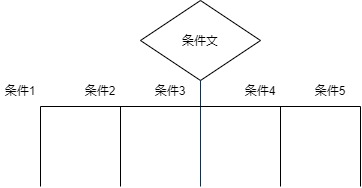
## 課題2

以下のフローチャートとプログラムを改造してください。

改造内容：if文をswitch文に変えてください。



switch文は以下のように表現してください

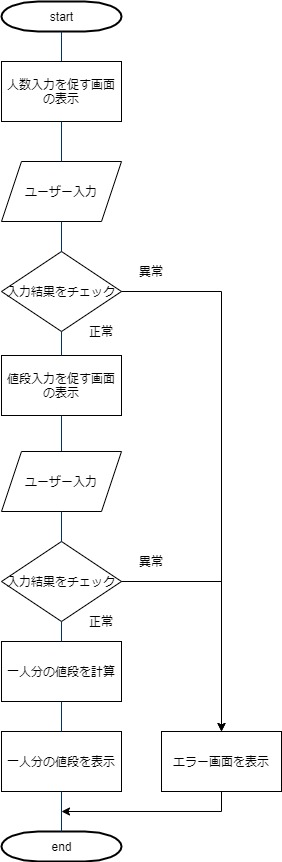


|  |
| --- |
| #include <stdio.h>  enum {  SIZE\_TYPE\_SHORT = 1,  SIZE\_TYPE\_TALL,  SIZE\_TYPE\_GRANDE,  SIZE\_TYPE\_VENTI,  SIZE\_TYPE\_MAX,  };  int main(void)  {  int size\_index = 0;  int scan\_ret = 0;  /\* 番号入力画面を表示 \*/  printf("[1]ショート（Short）\n");  printf("[2]トール（Tall）\n");  printf("[3]グランデ（Grande）\n");  printf("[4]ベンティ（Venti）\n");  printf("容量を知りたいサイズを番号で入力して下さい。\n");  printf(":");  /\* ユーザー入力 \*/  scan\_ret = scanf\_s("%d", &size\_index);  if ((scan\_ret == EOF) || (scan\_ret == 0) || (size\_index < 0)) {  printf("入力値が異常です\n");  } else {  /\* 結果表示 \*/  if (size\_index == SIZE\_TYPE\_SHORT) {  printf("ホット240ml/アイス300mlです\n");  } else if (size\_index == SIZE\_TYPE\_TALL) {  printf("350mlです\n");  } else if (size\_index == SIZE\_TYPE\_GRANDE) {  printf("470mlです\n");  } else if (size\_index == SIZE\_TYPE\_VENTI) {  printf("590mlです\n");  } else {  printf("そんなサイズはありません\n");  }  }  return 0;  } |

## 課題3

以下のフローチャートとプログラムを改造してください。

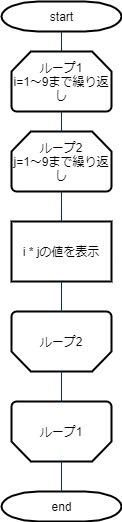
改造内容：余りを切り上げにして、おつりの値段も表示するようにしてください。



|  |
| --- |
| #include <stdio.h>  #include <errno.h>  int main(void)  {  long price = 0;  long count = 0;  long div\_ret = 0;  int scan\_ret = 0;  int result = -1;  do {  /\* 人数入力を促す画面の表示 \*/  printf("割り勘の計算をします。人数を入れてください\n");  printf(":");  /\* ユーザー入力 \*/  scan\_ret = scanf\_s("%ld", &count);  if ((scan\_ret == EOF) || (scan\_ret == 0) || (count <= 0)) {  break;  }  /\* 値段入力を促す画面の表示 \*/  printf("値段を入れてください\n");  printf(":");  /\* ユーザー入力 \*/  scan\_ret = scanf\_s("%ld", &price);  if ((scan\_ret == EOF) || (scan\_ret == 0) || (price <= 0)) {  break;  }  /\* 一人分の値段を計算 \*/  div\_ret = price / count;  result = 0;  } while (0);  /\* 結果の表示 \*/  if (result == 0) {  printf("一人分は%ld円です\n", div\_ret);  } else {  printf("エラーが発生しました\n");  }  return result;  } |

## 課題4

フローチャートの通りにプログラムを作成してください。



## 課題5

課題3のプログラムに出てきた「ユーザー入力」の処理を関数化しましょう。

## 課題6

以下のメンバを持つ構造体を作成し、ユーザー入力によって名前、テストの点数を10人分入力させて、名前・点数・評価を一覧表示するプログラムを作成しましょう。

|  |  |
| --- | --- |
| 名前 | 最大２０文字 |
| テストの点数 | 0～100の数字 |
| 評価 | A：テストの点数が90～100点  B：テストの点数が60～89点  C：テストの点数が30～59点  D：テストの点数が29点以下 |