計算機言語 I 第 10 回
関数 (その 2)

この講義資料は, 次の場所にあります.

```
http://www.math.u-ryukyu.ac.jp/~suga/gengo/2018-1/10.pdf
```

1 エラーと警告

コンパイラを実行した時に, 問題がなければ何もメッセージが出ませんが, 問題がある時には, エラー (error) と警告 (warning) の 2 種類のメッセージが出ます. これらの違いは, 次です[*1].

エラー (error) ソースファイルの問題が理由で, コンパイラの実行が最後までできない. この場合, 実行ファイル a.out はできません.

警告 (warning) ソースファイルには問題があるが, コンパイラの実行は最後まで終わっている. この場合, 実行ファイル a.out は作られています. ただし, ソースファイルに問題があるため, a.out の実行時に何らかの問題が起こる可能性があります. C 言語で特に注意すべきは, 配列の範囲外へのアクセスです. C 言語では, そのようなソースファイルをコンパイルすると, エラーではなく, 警告 (warning) になり, 実行ファイルが作られてしまいます. このような実行ファイルを実行すると, 実行時にエラーになります. また, C 言語は, OS の基本的な部分を作る際のプログラミング言語として使われていますが, これが理由で, セキュリティーホール (security hole) がシステムの中に含まれるということが, 起こっています.

2 ライブラリ関数と man コマンド

C は Linux のモデルとなった Unix の開発用言語として作られました. Unix 系の OS では, C 言語の開発環境が付いている場合が, 多くあります. 開発に際して, ライブラリ関数の仕様を知りたくなる要求が, 当然のあるわけですが, それは, man コマンド (オンラインマニュアル) で読むことができます. ちなみに, 昔の Unix では, man コマンドの出力を印刷した紙のマニュアルが付いていました.

計算機概論 I で man コマンドを紹介しましたが, ライブラリ関数は, (紙) マニュアルの第 3 節にあります. そこで, man コマンドの 2 番目の引数に, 3 という節番号を付け加えると, ライブラリ関数の記述を読むことができます.

```
man 3 printf
```

[*1] 言葉が違うということは, 起こっていることも違うという当たり前のことに注意してください

やってみると分かりますが, printf に類似した関数がたくさんあって, それらに対する記述 (仕様や使い方) を読むことができます.

3 関数呼び出しでの動作

C 言語での関数呼び出しでの動作のモデルとなる考え方を提示しておきます. 現実の動作では, ここに述べたようには実行されないことも多いですが, プログラマには, ここで述べるようなモデルで動作するように見せます.

例として, 次のようなプログラムの動作を考えます.

```
int add(int x, int y)
{
    return x+y;
}

int main(void)
{
    int a=3;
    int b=5;
    int c;

    c=add(a,b);
}
```

このプログラムが, 実行時にどのような動作になるかを解説します. まず, main から実行が始まることに注意します.

1. 変数 a の保持する場所をメモリに確保し 3 を代入する.
2. 変数 b の保持する場所をメモリに確保し 5 を代入する.
3. 変数 c の保持する場所をメモリに確保する.
4. 関数呼び出し add が次のように実行される.
   (a) add の 2 つの引数 x, y のための場所をメモリに確保する.
   (b) x に a の値 3, y に b の値 5 をコピーする.
   (c) add の返り値を代入する c の場所と上で確保した x, y の場所の情報を保持して, 関数 add に飛ぶ.
   (d) add の処理結果を, 上で保持している c の場所にコピーして, main に戻る.

上の (c) を見ればわかるように, 関数呼び出しは, 特定の場所へ処理を飛ばす (ジャンプする) ことと同じです. 条件分岐のジャンプとの違いは, その上の (a), (b) の処理が必要なところです. このあたりが, 関数呼び出しのオーバーヘッド (余計な動作) に関係する部分です. ただし, 上の動作は, あくまでも C 言語の処理系がプロブラマに見せている計算機のモデルであって, 実際には, 上の (a), (b) の動作を全くしなくても, そのように (プログラマには) 見える CPU も開発されています. したがって,「プログラムを関数に分割して記述する」ときのオーバーヘッドは, 今では, ほとんど気にされません. (全く無いわけではない).

上の動作からわかる 注意を述べておきます.

値渡し (Call by value)

関数に値を代入する際に, 代入される引数に元の引数の値がコピーされます (上の (a)). このように, 元の値のコピーが代入される処理を, 値渡し (call by value) と言います. 処理の仕方からわかるように, 関数において, 引数で受け取った元の値を変化させることは, そのままではできません.

「値渡し」ですが, 配列変数だけは例外 (配列のコピーは大変) 扱いになっています. これについては, 後の講義で述べます.

コピーでは無く, 元の値そのものを渡すようなプログラミング言語もあります. そのような処理は, 参照渡し (Call by reference) と言います.

最近では, 値渡しを実装するのが, プログラム言語での標準となっています. その主な理由は, 次です.

- プログラムは,「変数値を変更することにより所定の処理を実現する」ものである.
- 特定の値が, プログラムの多くの場所で変更できるようになっていると, 人間がそのプログラムの処理を理解するのが大変になる.

仮引数名は何でも良い. 無くても良い

上の (a) を見ればわかるように, 引数の情報として必要なのは, どのような型の変数かという情報です. したがって, 仮引数の変数名 (上の例だと, x, y) はどのような名前を用いても構いませんし, x, y を省略することもできます.

ただし, 現実的には, 引数の意味を表す変数名をつけるのが, 読みやすいプログラムになると思います.

4 値を返さない関数, 引数を取らない関数

今までのプログラムで,

```
int main(void)
```

と書いてきました. この void というキーワードは, C 言語の色々な場所で現れるのですが, 関数の引数部分に書くと,「引数を持たない関数」という意味になります.

「何らかの処理だけをして値を返さない関数」というものも考えられます. (他の言語では, このようなものは関数と呼ばず, 手続き (procedure) と呼ぶことが多い.) このような関数は,

```
void hogehoge(int i, double x)
...
```

と返り値キーワードを void にします.

古い C の本では

```
main()
```

という形で main を記述している本が多数あります. これは, C 言語の仕様として,

返り値を省略した関数の返り値は int 型である.

というのがあったためです.(今でも過去に対する互換性のためにある.) ただし, 現在では, このような関数を記述すると warning が出ます.

引数部分の void は省略しても, 全く問題ありません. (わざわざ void を記述しなければならない場合は, あまり無い.)

5 名前 (識別子) の有効範囲

以前の講義でも述べましたが,「変数名 (識別子) の有効範囲は, ブロックの中だけ」です.

プログラミングでは, 変数の値を名前で参照します. その名前による参照がどの範囲で可能か? ということが問題になります. そして, その回答は, ブロック内だけということです.

長いプログラムになると, 同じ変数名 (意識的にも無意識的にも) を使うことがあります. その時に,「別のブロックにある同じ変数名を持つ変数は, 全く無関係である.」が成立するように, C の言語仕様は定められています. 従って, 異なる 2 つの関数が同じ変数名を持っても, それらは全く関係のないものになります.

同じ名前 (識別子) でも, 関数名については, 変数名と違う扱いになります. これについては, 後の講義で述べます.

6 プロトタイプ宣言

長いプログラムでは,「main から記述を始めて, 後でそこで用いる関数の記述をする.」という記述が通常です. (トップダウン方式で, プログラムを書いていく.)

前回述べたように, 関数は,「それを使う前にその関数の引数や返り値の情報をコンパイラが知る必要がある.」のです. コンパイラは, 関数が適切に利用されているかをこの情報をもとにチェックします.

上のことを実現するために, main の前に関数の形を記述しておきます. これを「関数のプロトタイプ宣言」と言います.

最初に例示したプログラムだと, 次のような記述になります. main の前にある部分が, add のプロトタイプ宣言です. 下の方にある関数 add の実体は,「関数定義」と呼ばれます.

下のようなプログラムを書く時には,

1. まず main を書く.
2. main の前に add のプロトタイプ宣言を書く.
3. add を書く.

のような順で書いていきます. 実際には, 上に行ったり, 下に行ったりを繰り返して記述していきます.

```
int add(int x, int y);

int main(void)
{
    int a=3;
    int b=5;
    int c;

    c=add(a,b);
}

int add(int x, int y)
{
    return x+y;
}
```

7 本日の実習

教科書を読みつつ, p. 137, List 6-2 から p. 146, List 6-10 までを実行せよ. また, p. 149 まで教科書を読むこと. そのあとに教科書で出てくる, 関数における配列の受け渡しについては, 次週の講義で少し詳しく解説します.

教科書では,

```
while (n-- > 0)
    putchar('*');
```

のような, 後置デクリメントと比較演算を同時にやるプログラムが書かれていますが, これは,

```
while ( n > 0 ){
   putchar('*");
   n--;
}
```

とわかりやすく書き直してください.

以前にも述べましたが, 上の 2 つの記述で, 今のコンパイラは, 全く同じ機械語を生成します. 従って, 人間が理解しやすい形を記述します. また, 上の 2 つが同じ動作である (後置デクリメントの仕様) ということが理解できるようになってください.

レポート問題

締め切りは 6 月 25 日.

- 配列の範囲外をアクセスするプログラムをいろいろ書いて, 警告は出るものの a.out ができるものを実行せよ. 実行時エラーが起きるかどうかを確かめ, 実行時には, どのようなエラーが起こるかをレポートせよ. 件名: warning