プログラミング演習 A 教材 (#3)

■ コンパイル

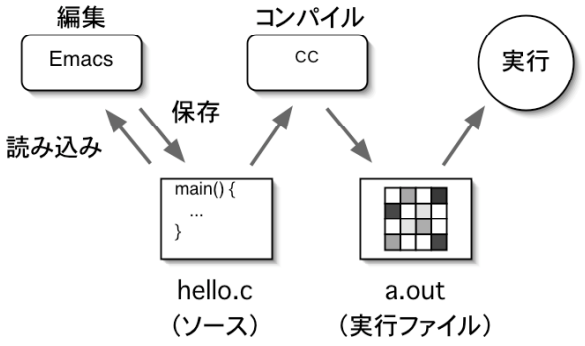
C プログラムはコンパイルという処理を経ないとコンピュータに実行させることができません。

□ コンパイルという変換過程

多くのコンピュータ・システムは、人間が書いたプログラムを直接実行することができません(*1)。C プログラムをコンピュータに実行させるためには、それをコンピュータが実行可能な形式に変換してから実行するというステップを踏みます。この変換のことをコンパイルと呼んでいます。

右図はそれを模式的に書いたものです。

1. たとえば hello.c という C プログラムを Emacs で編集、つまりディスクから読み込んで修正し、ディスクに保存し直していたとします。
2. プログラムができあがれば、それを Emacs で保存し、
3. できあがった hello.c ファイルをコンパイルして a.out というファイルに変換します。
   % cc hello.c とします。
4. できあがった a.out ファイルを、./a.out （先頭にピリオドとスラッシュをつける）として実行します。



こうした関係にあるとき、hello.c をソース（またはソースプログラム）、a.out を実行ファイル（またはオブジェクトプログラム）と呼びます。

*1 多くのコンピュータ・システムは、人間が書いたプログラムを直接実行することができません。ただこれは単にコンピュータが直接実行できるプログラムを人間が書くのは面倒なので、書きやすいプログラミング言語でプログラムを書き、それをハードウェアが直接実行できる言語に機械変換するという方法を多くのケースで採っているというだけのことです。つまり便利さの問題です。原理的には人間が書けるプログラムを直接実行できるハードウェアが作れないわけでも、ハードウェアが直接実行できるプログラムを人間が書けないわけでもありません。

□ コンパイル・エラー

コンパイルは常に成功するとは限りません。例えばプログラムに文法上の誤りがあった場合、コンパイルは中断され、エラーメッセージが表示されます。このとき a.out 実行ファイルは作成されません。

例えば文法上のエラーのあるプログラムをコンパイルした例を示します。(上がソース、下がエラーメッセージ)

```
#include <stdio.h>
main(){
  pritntf("Hello World!")
}
```

```
hello.c: In function `main':
  ★ hello.c の ‘main’ 関数の中で、
hello.c:3: parse error before `}'
  ★ 3 行め：`}'の前が解釈できない
```

ソースの誤りは 3 行目の「printf( )」の最後に「；」（セミコロン）を付け忘れたことです。エラーメッセージを読むと、そのことが簡単な英語で書かれています。実際には表示されませんが、今回の例では簡単な和訳を★に続けてつけてみました。

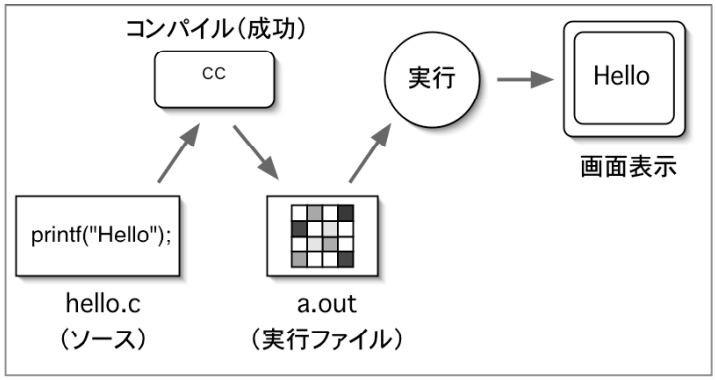
□ コンパイルに失敗した場合の a.out

このようにコンパイルに失敗した場合、a.out 実行ファイルは作成されませんが、削除もされません。つまりはじめてコンパイルした時に失敗しても a.out は作られませんが、以前にコンパイルに成功して a.out が一旦作られていた場合、最後のコンパイルが失敗しても a.out は削除されず、以前に成功してできたものが残っています。

コンパイルエラーの有無をよく見ないでプログラムを修正していると、エラーを起こしているにもかかわらず、たまたま古い a.out 実行ファイルを実行し、プログラムは修正した筈なのに結果が変わっていない、何故なんだろう？というように誤解する場合があります。

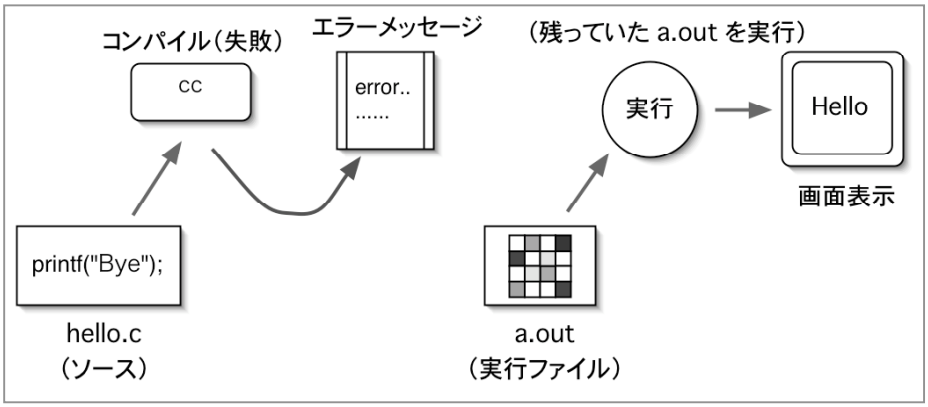
例えば Hello と画面に表示するプログラムを作っていたとします。

うまくできてコンパイルに成功し、実行して画面表示を確認しました。(右図)



このあと、Hello ではなく Bye と画面に表示するようにプログラムを修正しました。ところが修正を失敗して、コンパイルエラーが出てしまいました。(右図)

このとき a.out は書き換えられていません。たまたま前回作成された a.out が残っているだけです。



これではいくら実行しても画面表示は Hello のままです。何度 Hello を Hi や Yahoo に直してコンパイルし直しても、実行結果（画面表示）は変わりません。まずコンパイルエラーの原因を直す必要があります。

□ ワーニング

はじめのうちは遭遇しないと思いますが、重大な誤りではないため、warning （警告）だけで実行ファイルの作成まではする、というエラーもあります。以下のように先頭に warning と出ます。

```
sample.c:7: warning: assignment makes integer from pointer without a cast
```

理解して欲しいこと：
コンパイルエラーは必ずその原因を直して下さい。warning はともかく、それ以外の全てのコンパイルエラーを修正してからでないと、プログラムは一行も実行できないのです。

□ エラーメッセージの例

・引用符を閉じ忘れた場合のエラーメッセージ

printf( ) 関数の中には表示したい文字列を引用符で囲って指定します。その引用符を閉じ忘れた場合は、以下のようなエラーになります。

```
#include <stdio.h>
main(){
  pritntf("Hello World!);
}
```

```
hello.c:3: unterminated string or character constant
  ★ hello.c: 3 行目 : 終端処理がされていない文字列または文字定数あり
hello.c:3: possible real start of unterminated constant
  ★ hello.c: 3 行目 : 終端処理されていない定数が実際に開始されたのはこの行の可能性あり
```

行数（ 3 行目）を頼りに、何か文字列まわりで間違えたことをしていないか捜せば見つかるでしょう。

・関数の名前を間違えた場合のエラーメッセージ

例えば printf( ) 関数の名前を prittf( ) に間違えた時は以下のようなエラーになります。

```
#include <stdio.h>
main(){
  prittf("Hello World!");
}
```

```
/tmp/cc8DjRo7.o: In function `main':
  ★ 一時ファイル /tmp/cc8DjRo7.o で： main 関数の中で、
/tmp/cc8DjRo7.o(.text+0xf): undefined reference to `prittf'
  ★ 一時ファイル /tmp/cc8DjRo7.o で： prittf という未定義のものを参照している
collect2: ld returned 1 exit status
  ★ collect2: ld は 1 を終了ステイタスとして返したぞ（通常はゼロが返る）
```

少々わかりにくいエラーメッセージとなりましたが、関数の名前を間違えるとこのようになります。この場合は行数が表示されていませんので、prittf という名前をてがかりに捜すことになるでしょう。

経験則：
エラーメッセージは注意深く読む。
それでもエラーの場所が分からないときは最後に修正したところの周辺を疑うべき。

・複雑なケース

（この例は最初はあまり気にしないで下さい。多少なりとも複雑なプログラムを書くようになったら遭遇するケースとしていつかまた見直して下さい。）

以下のようなプログラム（ 1 から 10 までの合計を表示する）があります。

```
#include <stdio.h>
int main() {
  int count,total;
  total=0;
  for(count = 1; count <= 10; count++) {
    total = total + 1;
  }
  printf("result=%d\n", total);
  return 0;
}
```

for() のブロック

main() のブロック

このプログラムの 7 行目の }; を何かの拍子に間違えて消してしまったとしましょう。そのとき、エラーメッセージはこのようになります。

```
total.c: In function `main':
  ★ total.c: の main 関数のなかで、
total.c:11: parse error at end of input
  ★ total.c: 11 行目:入力の最後として解釈できない部分あり（入力の最後とは思えない）
```

7 行目にエラーがあるはずなのに 11 行目、つまり文末（入力の最後）にエラーがある、と言われてしまいました。11 行目をいくら見てもこれではエラーの原因がわかりません。

これは 5 行目の for( ) ではじまった { } で囲まれたブロックが、本来 7 行目で終わるはずだったのに、それがなかったため 11 行目まで続いていると解釈された結果です。
ところがこのプログラム全体は 2 行目の main() ではじまった { によるブロックで囲われているはずだったのに、11 行目は 5 行目の for( ) の { とペアだと解釈されたため、その後ろにまだ 2 行目の main( ) の { とペアの } があるに違いない、とコンパイラは考えました。
しかし何もないので「11 行目まで来たが、これが最後とは思えない」というエラーが出てくるわけです。
「分からないときは最後に修正したところの周辺が怪しい」という鉄則をおぼえてください。

```
#include <stdio.h>
int main() {
  int count,total;
  total=0;
  for(count = 1; count <= 10; count++) {
    total = total + 1;

  printf("result=%d\n", total);
  return 0;
}
```

}; を消してしまった！

for() のブロックはここまでだな？

main() のブロックの終わりがない！