C プログラム : コンパイル･リンク (実行以前) の仕組み

- エラーや警告を無視しない
  - 何故出ているのか, 意味を理解する
  - 意味がわからなければ C 言語について知らないことがある可能性がある
  - 大量に出た場合は最初のほうが重要
- システムのライブラリについて文句を言われていると思ったら man で調べよ
  - 必要な #include 句
  - 必要なリンクオプション (-lm)
  - -lm とかはコマンドラインの後ろにつける

- gdb で segmentation fault の場所を突き止められるようになろう
- ライブラリ関数，特に入出力（ファイルやネットワーク）は「返り値が正常か」チェックする
  - 色々な理由で失敗する
  - 引数の渡し間違いとか

説明の目的

- 色々な警告やエラーに対して
  - #include せよ
  - -lm をつけよ
- などの「対処法」の「意味」をもう少しよく理解する
- C プログラムから実行可能ファイルを作る過程を少し理解することで見えてくる

C プログラムから実行可能ファイルへ

C プログラム
(例 : ini.c)

```
int foo()
{

}

main()
{
  foo();
  printf(...);
  sin(...);
}
```

gcc ini.c

「コンパイル」

実行可能ファイル
(例 : a.out)

XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX
XXX

実践的注意
警告 (warning) とエラー (error)

- エラー：
  - 割と根本的な問題で , これがあるとそもそも実行可能ファイルはできていない (実行可能ファイルが存在したらそれは前に作った残骸でしょう)
- 警告：
  - プログラムに問題がある可能性が高いが , 一応実行可能ファイルはできている

```
ini.c:2:  警告 : return type defaults to ‘int’
ini.c: In function ‘main’:
ini.c:3:  警告 : implicit declaration of function ‘atof’
ini.c:4: error: ‘FILE’ undeclared (first use in this function)
ini.c:4: error: (Each undeclared identifier is reported only once
ini.c:4: error: for each function it appears in.)
ini.c:4: error: ‘fp’ undeclared (first use in this function)
```

コンパイル～大雑把なイメージ

- 「関数」を対応するマシン命令の列に変換する
  - 実行の際それらがメモリ上の適当なアドレスに配置される
- 変換の際「他の関数を呼び出す文」は , その関数が配置されているアドレス (それがあるものと仮定して) への呼び出し (call; ジャンプ) 命令に変換される

ini.c

```
int foo()
{
  ...
}

int main()
{
  foo();
  printf(...);
  sin(...);
}
```

gcc ini.c

a.out

```
foo:
  ...
  ...
  ...

main:
  ...
  call foo:
  ...
  call printf:
  ...
  call sin:
  ...

printf:
  ...
  ...

sin:
  ...
  ...
```

コンパイルが成功する要件 1

- プログラムが使っている (= 呼び出している) 関数は , 「どこかで」その中身が定義されていなくてはならない
- これが満たされていないと ... (リンクエラー)

```
int main()
{
  foo();
}
```

gcc ini.c

```
/tmp/cc2GxHhG.o: In function `main':
ini.c:(.text+0x12): undefined reference to `foo'
```

foo への「定義されていない」参照

(ましな翻訳) foo を使っている (参照している) のに , その中身 (定義) がどこにも見当たらない

「どこかで」とはどこのことか？

- 正にコンパイルした C プログラムの中 (foo)
- システムがあらかじめ用意しており , 自動的に検索されるライブラリファイルの中
  - printf, atoi,atof, などが納められている
- その他のライブラリファイルの中 (自動的には検索されない)
  - どのライブラリファイルを使うかを自分で (gcc のコマンドラインで) 指定する必要がある
    - -lm の意味
    - sin, cos, log, exp などを納めたライブラリ (libm.so) を使うことを指定している

コンパイルが成功する要件 2

- 「基本的には」プログラム中である関数 *f* が呼び出される場所より前で , *f* の引数や返り値の型 (type; 種類 ; int か , double か , 構造体か , など) が判明していなくてはならない
  - 「前」の意味 : プログラムの字面上で現れる「前」という意味

問題なし

```
double f ()
{
  ...
}

int main()
{
  f ();
}
```

問題あり

```
int main()
{
  f ();
}

double f ()
{
  ...
}
```

問題なし

```
double f ();
int main()
{
  f ();
}

double f ()
{
  ...
}
```

f の引数や返り値の型が分かっているようにする二つの方法

- 使う前に「定義 (definition) 」を書く
- 使う前に「宣言 (declaration) 」を書く
  - 中身を書かずに型だけを書いておく
  - 自分が書いたわけではない関数 (= システムが提供する関数 : printf, atof, sin, cos, etc.) の場合これが唯一の方法

```
double f () /*  定義  */
{
  ...
}

int main()
{
  f ();
}
```

```
double sin (double); /*  宣言  */
int main()
{
  sin (3.1);
}
```

#include の意味

- #include <stdlib.h> とすると , stdio.h というファイルが , コンパイル中のファイルに取り込まれる (copy&paste されていると思えばよい)
  - stdlib.h 内では atoi, atof などの関数が宣言されている
  - double atof(char *);
  - int atoi(char *);
  - ...

ただしここからが少しややこしい

- この規則を破った場合 ,
  - コンパイルが「エラー」になる場合もある
  - 「エラー」が出ずに , 「警告」が出る場合もある
  - 警告が出て , 実行時の挙動がおかしくなることがある
    - 引数や返り値が正しく受け渡されない (sin, cos, etc.)
  - 結果オーライで問題ない場合もある (printf, atoi, etc.)
    - これが意外と多いので逆に問題となるケースでハマる
- 以下を守れば大丈夫 (だがあえてもう少し深入りする)

C プログラム内に適切な #include を入れる
gcc に適切な ” -l なんとか” (この演習では -lm だけ) を与える

C コンパイラの規則

- 関数 *f* の定義･宣言より (プログラムの字面上) 先に *f* が呼び出されるのを見たとき
  - その場でエラーとはしない
  - *f* の返り値は int，引数は呼び出し時のものであると仮定する (implicit declaration)
  - gcc では -Wall をつけると以下の警告が出る

```
ini.c:3:  警告 : implicit declaration of function ‘foo’
```

C コンパイラの規則

- その後 *f* が定義されたとき
  - そこで定義された返り値や引数の型が以前の ” implicit declaration” と合致していれば OK
  - 合致していなければエラー
- その後 *f* が定義されなかったとき
  - 注： ありうる． システムが提供する関数だったり， 別の C ファイルで定義されている関数だった時
  - 素通し (整合性がチェックされないまま使われることになる． 実行は出きるが， 引数や返り値が正しく受け渡されない ⇒ わけの分からない挙動)

コンパイルが成功する要件 3

- (関数の呼び出しではなく), 変数や型の名前の使用は , 使用に先立って定義･宣言されていなければ即エラー
  - FILE * fp = fopen(...);
- システムが定義しているのであれば , 適切な #include を入れることで解決 (宣言してくれる)

```
int main()
{
  FILE * fp = fopen("foo", "rb");
  printf("%d\n", x);
}
```

```
ini.c:3: error: ‘FILE’ undeclared (first use in this function)
ini.c:3: error: (Each undeclared identifier is reported only once
ini.c:3: error: for each function it appears in.)
ini.c:3: error: ‘fp’ undeclared (first use in this function)
ini.c:4:  警告 : incompatible implicit declaration of built-in function ‘printf’
ini.c:4: error: ‘x’ undeclared (first use in this function)
```

第一日の課題ヒント

- 最初の数問 (00,01,02) はここであげた種類のエラーに対して ,
  - -lm をつける (リンクエラー)
  - #include を入れる
- ことが実際の修正の大部分を占める
- このスライドの話を理解するとそれぞれのエラーメッセージの意味もわかるようになる (と期待する)