コンピュータとプログラミング C言語編

1.イントロダクション

阿部洋丈 / ABE Hirotake
habe@cs.tsukuba.ac.jp

この授業の目的

- アセンブリ言語と高級言語の中間に位置する「低水準言語」の代表であるC言語の習得を通じ、コンピュータの動作についての深い理解を得る。
- 機械語レベルの知識を踏まえた上で、効率的なプログラムを書けるようになる。
- C言語で発生しやすいバグを理解し、バグの少ないプログラムを書けるようになる。

C言語

- 低水準の記述が可能な汎用プログラミング言語の一つ
  - 低水準ゆえに「高級アセンブラ」と揶揄されることも
- 1972年に AT&T Bell Labs の Dennis Ritchie (dmr) が設計
- IEEE Spectrum 誌の人気ランキング (2024年)では第9位 (前年は4位)
  - 1位はPythonで、その後に Java, JavaScript, C++, TypeScript, SQL, C#, Go, C, HTML, Rust, Mathematica, PHP, Shell, Lua, SAS, …と続く

Archetypal hackers ken (left) and dmr (right)
(http://www.catb.org/~esr/jargon/より抜粋)

なぜC言語？

- 古くてもまだまだ現役
  - Linux 等のオペレーティングシステムの実装
  - Python や Ruby など、他の言語処理系の実装
  - 組込制御デバイスやIoTデバイスでの利用
- 低水準ゆえのメリットがある
  - メモリやハードウェアを直接操作するプログラムの記述が容易
  - 実行時のオーバーヘッドが少ない
- 過去の膨大な遺産が蓄積されている

Pythonとの主な違い

- C言語はコンパイラ型言語（Pythonはインタプリタ型）
- C言語にはガベージコレクションが無い
- C言語にはポインタ (pointer)がある
  - 一言で言えば「アドレス変数＋α」
  - Pythonには、オブジェクト参照はあるが、ポインタ相当の機能は無い
  - ポインタについては第4回で詳しく説明

この授業の設計

- 以下の教科書の内容をベースに、より発展的な内容を補いながら実施
  - 「未来へつなぐデジタルシリーズ 30 C言語」（共立出版）
- 授業で使用するスライドはすべて manaba 上で配布
- 教科書を購入する必要は無いが、基礎部分の内容を補いたい場合は購入検討を薦める

補足： C言語の仕様について

- C言語は、その長い歴史故に、さまざまな仕様が存在
- 本授業では、GCC 4.x 系の default である「gnu89」に準拠
  - ISO 規格である C89 をベースとして、C99 の一部を取り入れたもの
    - 「//」 によるコメント行、変数宣言がブロックの先頭でなくても良い、etc.
    - C89 で書けば gnu89 にも準拠すると考えて良い
  - 対応している処理系が多い。また、Linux の大部分は gnu89 で記述
- GCC 以外のコンパイラを使う場合は若干の注意が必要
  - clang は -std=gnu89 で対応可能。ただし、大抵はこれ無しでも OK

Hello world --- 最初のプログラム

- C言語について解説した最初の書籍 "The C Programming Language" (通称 K&R) で最初に登場するプログラム
  - 実際にはそれ以前（BCPL の頃）から使われていたらしい
- そこから転じて、C言語以外を習得する場合でも、初心者が最初に書くプログラムとして一般的に
- 画面に "Hello, world" と一行表示するだけのプログラム

Hello world（教科書より）

```
#include <stdio.h>


int main(void) {
  printf("Hello World ! \n");
  return 0;
}
```

Hello world の走らせ方

```
$ vi hello.c
$ gcc hello.c -o hello
$./hello
Hello World !
$
```

コンパイル
(実行可能
ファイルの生成)

コンパイル結果
の実行

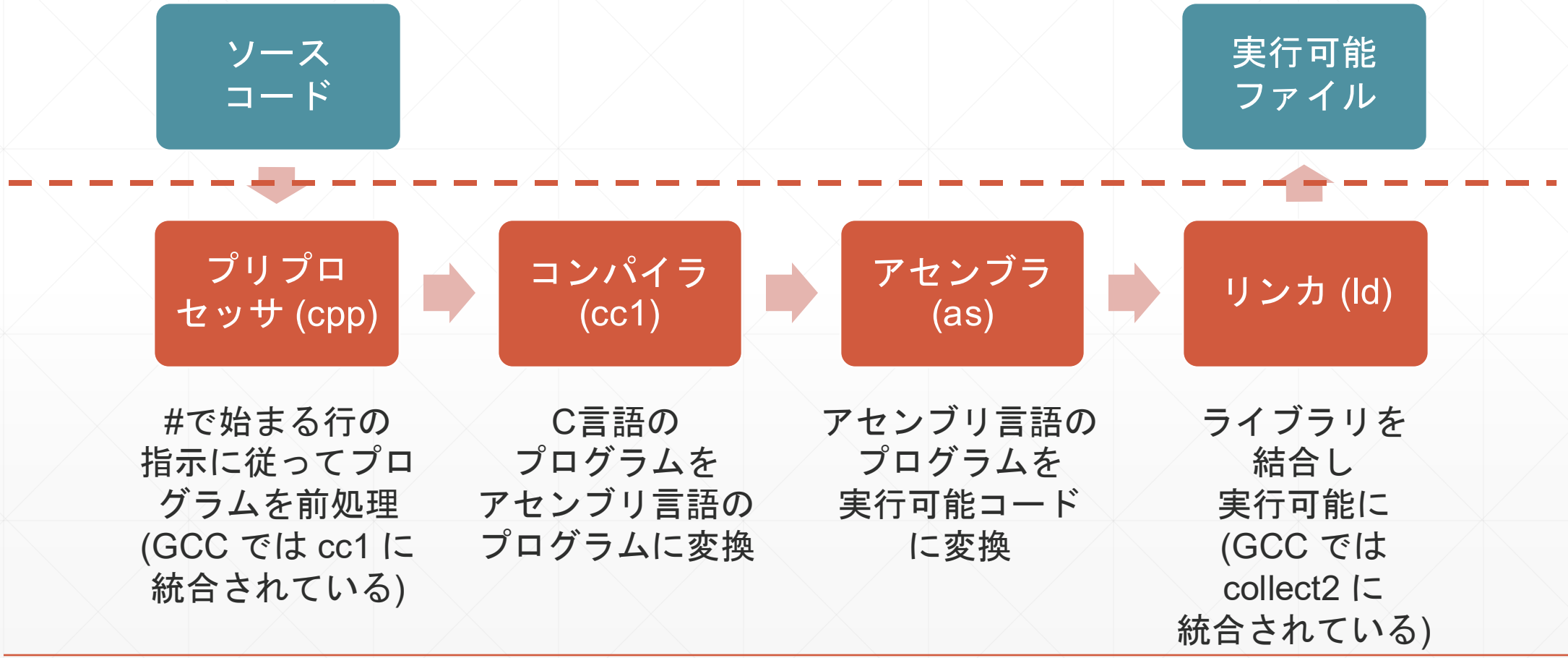
出力結果

"gcc hello.c -o hello" の裏側

```
$ gcc –v hello.c -o hello
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,java,fortran,ada,go,lto --enable-plugin --enable-initfini-array --disable-libgcj --with-isl=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/isl-install --with-cloog=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-install --enable-gnu-indirect-function --with-tune=generic --with-arch_32=x86-64 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)
COLLECT_GCC_OPTIONS='-v' '-o' 'hello' '-mtune=generic' '-march=x86-64'
 /usr/libexec/gcc/x86_64-redhat-linux/4.8.5/cc1 -quiet -v hello.c -quiet -dumpbase hello.c -mtune=generic -march=x86-64 -auxbase hello -version -o /tmp/cc8j1fGS.s
GNU C (GCC) version 4.8.5 20150623 (Red Hat 4.8.5-39) (x86_64-redhat-linux)
        compiled by GNU C version 4.8.5 20150623 (Red Hat 4.8.5-39), GMP version 6.0.0, MPFR version 3.1.1, MPC version 1.0.1
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/lib/gcc/x86_64-redhat-linux/4.8.5/include-fixed"
ignoring nonexistent directory "/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../x86_64-redhat-linux/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/x86_64-redhat-linux/4.8.5/include
 /usr/local/include
 /usr/include
End of search list.
GNU C (GCC) version 4.8.5 20150623 (Red Hat 4.8.5-39) (x86_64-redhat-linux)
        compiled by GNU C version 4.8.5 20150623 (Red Hat 4.8.5-39), GMP version 6.0.0, MPFR version 3.1.1, MPC version 1.0.1
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: d8f4c208bcaf7e279b70f7290eda3265
COLLECT_GCC_OPTIONS='-v' '-o' 'hello' '-mtune=generic' '-march=x86-64'
 as -v --64 -o /tmp/ccLhx4NK.o /tmp/cc8j1fGS.s
GNU assembler version 2.27 (x86_64-redhat-linux) using BFD version version 2.27-43.base.el7
COMPILER_PATH=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/:/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/:/usr/libexec/gcc/x86_64-redhat-linux/:/usr/lib/gcc/x86_64-redhat-linux/4.8.5/:/usr/lib/gcc/x86_64-redhat-linux/
LIBRARY_PATH=/usr/lib/gcc/x86_64-redhat-linux/4.8.5/:/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/:/lib/../lib64/:/usr/lib/../lib64/:/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../:/lib/:/usr/lib/
COLLECT_GCC_OPTIONS='-v' '-o' 'hello' '-mtune=generic' '-march=x86-64'
 /usr/libexec/gcc/x86_64-redhat-linux/4.8.5/collect2 --build-id --no-add-needed --eh-frame-hdr --hash-style=gnu -m elf_x86_64 -dynamic-linker /lib64/ld-linux-x86-64.so.2 -o hello /usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/crt1.o /usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/crti.o /usr/lib/gcc/x86_64-redhat-linux/4.8.5/crtbegin.o -L/usr/lib/gcc/x86_64-redhat-linux/4.8.5 -L/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64 -L/lib/../lib64 -L/usr/lib/../lib64 -L/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../.. /tmp/ccLhx4NK.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s --no-as-needed /usr/lib/gcc/x86_64-redhat-linux/4.8.5/crtend.o /usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/crtn.o
```

gcc コマンドの裏で複数のプログラムが動いている

コンパイル処理の流れ


ソース
コード
実行可能
ファイル
プリプロ
セッサ (cpp)
コンパイラ
(cc1)
アセンブラ
(as)
リンカ (ld)
#で始まる行の
指示に従ってプロ
グラムを前処理
(GCC では cc1 に
統合されている)
C言語の
プログラムを
アセンブリ言語の
プログラムに変換
アセンブリ言語の
プログラムを
実行可能コード
に変換
ライブラリを
結合し
実行可能に
(GCC では
collect2 に
統合されている)

hello.c を受け取ったコンパイラの気持ち

- main という関数を生成する必要があるようだ
  - 引数は無し(void)、返戻値は int 型
  - printf という関数を呼んで、それが終わったら（printf の結果に関係なく）整数 0 を返せば良いらしい
- ところで、printf って、何？
  - 関数っぽい書式だけど、だとしたら一体どんな関数？ 引数の数や型は？ 返戻値の型は？
  - もし本当に関数だとしたら、その実体はどこにある？ どうすれば呼べる？
  - もしかしたら関数じゃない可能性も…？ （マクロ（後述））

プリプロセッサ (cpp)

- "#" で始まる行に従ってプログラムの前処理を行う
  - #include で指定されたヘッダファイル (stdio.h) の取り込み
  - #define で定義されたマクロの置換（後述）
  - 他にもあるが、ひとまずはこの二つを覚えて おけばOK
- stdio.h とは
  - 標準入出力 (standard input/output) を扱うための様々な定義を集めたヘッダファイル。printf を使うための定義もここに入っている
  - コンソールに文字を表示したりファイルを読み書きする時に使う
- gcc に -E オプションを与えると、cpp の結果が画面に出力される

コンパイラ (cc1)

- C言語で書かれたプログラムをアセンブリ言語のプログラムに変換
  - 対象となるアーキテクチャ毎に必要
  - VMコードに変換した後で更に各アーキテクチャ用に変換するコンパイラもある
- コンパイラの中身について詳しく知りたい人は「プログラム言語処理」(GB31301)の受講を推奨
  - 字句解析、構文解析、コード生成、コード最適化、etc.

アセンブラ (as)

- アセンブリ言語で書かれたプログラムを機械語に変換
- 既によく知っているはずなので省略
- gcc に -S オプションを与えると、アセンブラによる処理を省略し、機械語コードの代わりにアセンブリコードが出力される。
  - "gcc -S hello.c" -> hello.s というファイルが得られる

リンカ (ld)

- 実は、as の出力結果はそのままでは実行できない。実行できるようにするためには以下の二つが必要
  - printf 関数の機械語コードを含むライブラリ (libc)
  - 環境を整えた上で main 関数を呼び出す機械語コード (crt0)
- リンカは、as の出力と上記のコードをリンクして実行可能プログラムを構築する
  - リンクとは、つまりコール先のアドレスの解決

マクロとは

- cc1 にプログラムを渡す前に文字列置換を行う機能
- "#define" を使ってマクロを定義しておくと、cpp がそれに従って自動的に処理
- 2種類のマクロ
  - オブジェクト形式マクロ（引数を取らない定数）

    #define MAX 10
  - 関数形式マクロ（引数を取る）

    #define ADD(x,y) ((x) + (y))
- 関数形式はバグの原因に比較的なりやすいので注意が必要

オブジェクト形式マクロを使う理由

- 可読性の向上
  - マクロを使わない場合： if (i > 10) error();
  - マクロを使う場合： if (i > MAX) error();
- 保守性の向上
  - MAX の値が 10 から 20 に変更になった場合、
    - マクロを使っていれば、#define の行だけを修正すれば良い
    - マクロを使っていない場合は、10が出てくる箇所を見つけ、それが MAX の意味で使われているのかを逐一判断する必要あり
- 処理系固有の定数の参照
  - #include <limits.h> をすると、INT_MAX などのマクロが使える

今週の課題（肩慣らし；レポートなし）

- Hello world プログラムを入力し、コンパイルし、実行せよ。
- Hello world プログラムのソースコードを cpp や cc1 で処理した結果を確認せよ
  - printf を使うための定義がどのように展開されているか(cpp)
  - printf 関数を呼び出すためのアセンブリコードがどうなっているか(cc1)