計算機言語 I 第 12 回
整数型

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/2018-1/12.pdf`

1 ヘッダファイル

```
#include <stdio.h>
```

の意味を正確に説明します.

- `#` で始まっているので, cpp に対する指示である.
- `#include` は指定されたファイルをソースに含めるという意味である.
- ソースに含めるファイルの名前は, stdio.h というファイルである.
- `<stdio.h>`の両端の不等号`<` と `>` は, stdio.h が, C コンパイラの処理系が標準的に定めてあるディレクトリにあることを意味する.
- stdio = starndard input output (標準入出力) の略.
- ファイル拡張子の .h はヘッダファイルの意味で, stdio.h には標準入出力を利用する際に用いるライブラリ関数のプロトタイプ宣言や, 定数宣言が書かれている.

上の「コンパイラの処理系が標準的に定めてあるディレクトリ」ですが, Linux や MacOX X のような Unix 由来のシステムでは, `/usr/include` です. ここには, stdio.h 以外にも他のライブラリ関数のヘッダファイルや, 処理系の仕様から定まる色々な定数を記述したヘッダファイルがあります. Gnome 端末で次を実行してみてください.

```
ls /usr/include
less /usr/include/stdio.h
```

2 基本変数型での数

教科書第 7 章の内容は, 基本的に 20 年以上前のもので, 現在では大きくわかった部分もありますので, ここで, 現代風に書き換えておきます.

コンピュータ内のデータ

コンピュータ内のデータは, 基本的に 2 進法の数です. これは, コンピュータという機械を作るにあたって,「工学的に合理的である」, すなわち,「現在の技術で性能の高い自動計算機を作りやすい」が理由になっています. 2 進法以外の数でデータ表現をすることや, 電子式以外の方式なども, 黎明期には考えられたようですが, 今の形式が工学的に成功しました.

ときどき報道などに現れる「量子コンピュータ」なるものが実用化されるなら,「データ=2 進法の数」という等式は崩れるかもしれませんが, 現時点では, 量子コンピュータがどの程度実用化まで持っていけるかは, わかりません.

さて, C における基本変数型のうち, `char` と `int` は整数型と呼ばれています. これらは, 共に整数値を保持するためのものです. ただし, どの範囲の値が保持できるのか (整数型変数のビット幅) という部分は, 処理系依存で言語仕様にはありません (処理系依存であるというのが言語仕様).

`char` 型には,

- `unsigned` (符号無), `signed` (符号付)

`int` 型には, 上の 2 つに続けて,

- `short` (短い, 範囲が狭くなる)
- `long` (長い, 範囲が広くなる)
- `long long` (より長い, 範囲が long 以上)

の言葉を前につけることができます. 例えば, `unsigned short int` は符号無しの短い整数型という意味です. これらの整数型に対しては, 次の大小関係は規格で定められています.

$$\begin{aligned}\texttt{char}\text{ 型のビット幅} &\leq \texttt{short int}\text{ 型のビット幅} \leq \texttt{int}\text{ 型のビット幅}\\ &\leq \texttt{long int}\text{ 型のビット幅} \leq \texttt{long long int}\text{ 型のビット幅}\end{aligned}$$

ここで注意して欲しいのは,「上の不等式は, 等号が成立することが多い.」です.

上のようになっている理由は, C の開発時期と開発目的が背景にあります.

コンピュータの心臓部 CPU には, レジスタ (register) と呼ばれる計算のためのデータを記憶する場所があるます. このレジスタのビット幅で, xx ビット CPU と呼ばれます. このレジスタのビット幅は, CPU が最も効率よく処理できるデータサイズです. C の開発初期では, 16 ビットの CPU がターゲットマシンだったと思われます.

C は OS を作り上げるために作成された開発環境なので, 上の整数型は, 初期は CPU のビット幅に合わせていました. すなわち, `int` 型が CPU のビット幅に一致するようにコンパイラが作成されていたのです. そうすることにより, CPU の処理速度を稼ぐことができ, コンピュータを高速に動かすようにしたのです. `short` は, 当時はメモリ資源が貴重であったため, メモリ消費を小さくできるようにするために定義され, `long` は, `int` では範囲が狭すぎるときに利用されました.

C が作られて発展していく過程で, CPU のビット幅の主流が, 16 ビットから 32 ビットに変わっていきました. この際に, 過去のコードを変更することなく, また高速なプログラムが出来上がるように, 基本型のビッ

ト幅は具体的に記述せず, 上のような不等号だけを規定しただと考えられます. ビット幅が増える分には, 過去のプログラムのコンパイルは, 問題はあまり起きないからです.

その後, 現在のように 64 ビット CPU が主流になりました. 32 ビット整数より広い範囲の整数値も扱えるように, `long long` が制定されました. 現時点では, CPU は 64 ビットですが, `int` は 32 ビットであるコンパイラが多いようです. これは, 32 ビット CPU の時代が長かったからだと思いますが, 今後, `int` は 64 ビットになるのではないかと思います.

`char` 型は, 文字 1 文字分を保持するための変数型となっています. ただし, 歴史的な事情から, 多くの場合 Ascii コード 1 文字分です. Ascii コードは 7 ビットですから, 7 ビット 以上の記憶領域があれば良いのですが, 8 ビット が標準的な実装のようです. `signed`, `unsigned` は, `int` 型には意味が明白ですが, `char` 型は, 処理系依存です. すなわち, `char` を符号付きとして処理する処理系と符号無しとして処理する処理系があります. これも, 過去において, メモリ節約のため, 小さい値した取らない整数を保持するために `char` を利用した際に問題になったことで, 今書くコードでこれが問題になることは, ほとんどないと思います.

`signed`, `unsigned`, `short`, `long` の使い方とその短縮名については, 教科書 p. 175 を見てください. これに加えて, `long long` がありますが, これの短縮系は, `long` と同様です.

整数型変数の負の数 (2 の補数)

整数型変数における負の数は, 2 の補数という形で実現されています.

簡単のために 4 ビット 整数型で説明します. 考え方は 1111+0001 = 10000 という繰り上がり計算で, 4 ビット 整数だと 5 ビット目が捕まえられず, 0000 になることを利用します. すなわち, 4 ビット整数では, -1 は 1111 なのです. この方式では, 整数の符号を変える方法は, 各 ビット の 0, 1 を逆転させて (ビット反転と言います), 1 を加えます. 例えば, 0101 (10 進法で 5) の負の数は, $1010 + 0001 = 1011$ となります. 「ビット反転で -1 倍の数」という流儀もあり得るのですが (1 の補数), この場合, 0 を表すのが, 0000 と 1111 の 2 種類あることになるのが欠点です.

符号を考える整数計算では, 最上位のビットが 0 なら 正の数, 1 なら負の数と決め, 上で述べた 2 の補数を利用するのが標準的な実装のようです. 上の 4 ビット の例でいうと, 最大整数は $0111_2 = 7$, 最小整数は $1000_2 = -8$ となり, 0 に対して非対称は範囲になります.

型の時に述べた, `unsigned` は, 上の最上位ビットの符号を無視して正整数の計算をするという意味になります.

/usr/include/limits.h

処理系の整数型の数の範囲を記述してあるファイルです.

```
less /usr/include/limits.h
```

色々な定数の後に `U` や `L` がついていますが, その意味は, 教科書 p. 195 にあります. `LL` はそれから類推できるでしょう.

レポート問題

- 教科書 List 7-1 に `long long`, `unsigned long long` の結果も出力するように変更したプログラムを書け. 件名: List7-1.c

sizeof()

C には, プログラム内で使うことができる sizeof() というコンパイラ処理系が処理する関数があります. `char` 型に対する記憶領域のビット幅の比を与える関数です. sizeof の引数には変数型や式が入り, その型なり式の値が占めるメモリ幅の比が出力となります. ほとんどの処理系では, `char` 型が 1 バイトなので, その場合バイト数が出力されます.

教科書 p. 180, List 7-3 から p. 183, List 7-5 までを実行してください. List 7-3 では, long long, float, double, long double の 4 つの型のサイズも出力するように変更して実行してください. 同じサイズを持つものがたくさんあることを確認します.

整数型の数の表記, 8 進法, 16 進法

C のコンパイラは, 数字から始まる文字列は数と解釈するようになっています. その際, 0 から始まる文字列は 8 進法, 0x から始まる文字列は 16 進法の数表記として解釈します.

8 進表記は, Unix(Linux) だとファイルモード (2017 年度計算機概論 I, 第 6 回の講義) が 8 進で, readable = 4, writable =2, executable = 1 の和で表記されます (`rw-` = 4+2+0 =6, `rw-r--r--` = 644). 16 進表記は, IP(internet protocol) を扱うプログラムでは, ネットワークアドレスを扱うのに便利です. この辺にも, C が OS 開発プログラムとして, 発展した歴史が伺えます.

16 進表記での 10 から 15 を表す数字は, 大文字の A から F, 小文字の a から f のどちらでも大丈夫です.

$\texttt{0123} = 1 \times 8^2 + 2 \times 8^1 + 1 \times 8^0 = 81$
$\texttt{0xaf} = 10 \times 16^1 + 15 \times 16^0 = 175$

printf や scanf も 8 進法, 16 進法表記を扱うことができます.

レポート問題

- 16 進法の 1 桁の掛け算の表 (10 進法だと九九の表) を出力 (出力も 16 進表記) するプログラムを書け. 件名: FF.c

レポートの締め切りは, 7 月 9 日 (月)