ソフトウェア概論 A/B

-- ガイダンス/復習/代入 --

数学科 栗野 俊一

2012/09/28 ソフトウェア概論

伝言

私語は慎むように !!

- 教室に入ったら
  - 直に Note-PC の電源を入れておく
    - Network に接続し、当日の資料に目を通す
    - skype に Login する
    - Windows Update をしておこう
- やる気のある方へ
  - 今日の資料は、すでに上っています
    - どんどん、先に進んでかまいません
- 作業
  - Web 履修科目登録の確認
    - CST Portal も確認しておきましょう

前期の成績について

- 前期(ソフトウェア概論 A)の成績について
  - 成績処理が遅れてしまった
    - 再履の人は印刷が間に合わず、成績欄が空白に (ごめんなさい)
    - Web の方を参照してください
  - かなり甘く付けた
    - 後期は、辛くする予定..
    - レポート全提出+試験参加で、単位は保証する
- 評価内容に疑問がある場合
  - 講議終了後、栗野に申し出てください
    - ミスをしている可能性あり
  - メールでも構いません
    - kurino@math.cst.nihon-u.ac.jp

後期の方針

- 後期 (ソフトウェア概論 B) の方針
  - 基本は前期 (ソフトウェア概論 A) と同じ
  - ただし...
    - 前期の知識を仮定する : 身についていない所は復習する
    - 後期は前期を踏まえ、更に高度な内容になる予定
- 方針(ポイント)の復習
  - 私語厳禁：他人に迷惑をかけるな !!
    - 自分がやらないのは自分の問題(好きにすれば..)
    - 他人への迷惑は断固とした態度を取る
  - 実習重視：毎回 Note-PC /LAN を利用する
    - 習うより慣れろ / 普段から利用する
  - 評価：課題+試験(講議時間中に行う)
    - 前期より厳しく..
  - Web/Mail/Chat を「活用」する
    - 口を止めて、頭と手(目/耳)を動かせ

本日の予定

- 講義
  - ガイダンス
    - 前期と同じ (という事でほぼ終了)
  - 前期の復習
    - 前期の内容を概観 (これは、解っていると仮定される !!)
- 演習
  - 課題の提出

本日の課題 (2012/09/28)

- 先週の課題
  - なし (後期最初の講議なので)
- 今週 (2012/09/28) の課題
  - 課題 1:
    - ファイル名 : 20120928-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : 成績処理を行う
    - ファイル形式 : テキストファイル(C 言語プログラムファイル)
    - 引数で自分の成績(S,A,B,C,D)をいれて、順位を表示させてみよ
  - 課題 2:
    - ファイル名 : 20120928-2-QQQQ.c (QQQQ は学生番号)
    - 内容 : 論理演算による符号なし 8 bit 二進数の足し算
    - ファイル形式 : テキストファイル(C 言語プログラムファイル)

状況

- 前期の先頭で述べた事
  - 前後期で「C 言語の話」を内容を深めながら三回繰り返す予定だった
  - 一周目：プログラムを書くための C 言語で最低限の知識
    - これは済んだ
  - 二周目：C 言語の学習内容の大雑把な網羅
  - 三周目：C 言語を利用したプログラムミング
- 予定では、前期で、二回する予定だった
  - 実際には、二周目の途中でおわってしまった
- 二周目の残っている内容
  - 代入：すこし話したが、不充分
  - while 文：代入と組合せて利用する繰返し
  - 複雑なデータ構造：配列と構造体
    - これを、次の二三回で行う
    - 前期で単位が取れた人も、ここまでは聞いて欲い

前期の復習：プログラム

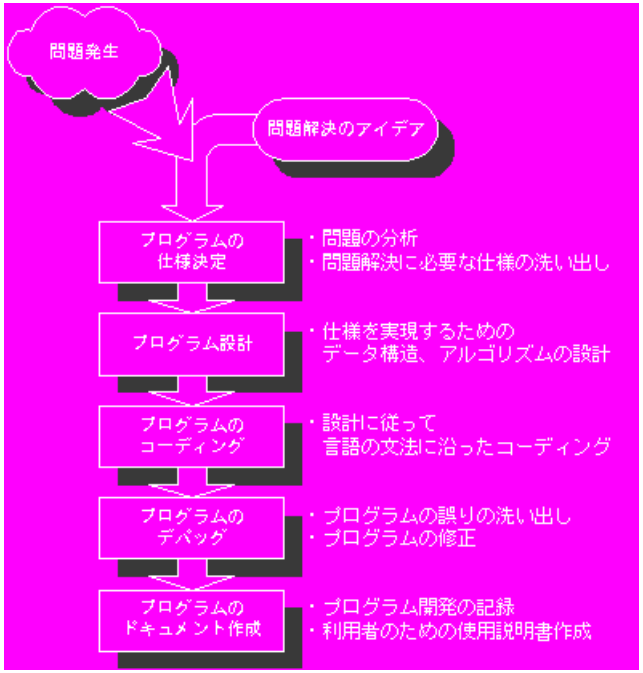
- プログラムとは
  - 命令を並べた物
    - 命令:計算機に何か(単純な事..)をさせる事ができる
    - 「どんな命令が利用できるか」は憶える必要がある
- 命令の組み合わせによって様々な機能(複雑な事)が実現できる
    - 「命令の組み合わせ方法」も考える必要がある
  - 計算機はプログラムによって動いている
    - プログラムに記述された命令を順番に実施しているだけ
    - 計算機の機能は即ちプログラムの機能
- プログラミングとは
  - 命令を自分の意図に従って並べる
    - 計算機に自分の意図通りの動作をさせる事ができる
  - 新しいプログラムを自分で作成する事ができる
- プログラミング言語とは
  - プログラムを記述するための言葉
    - 命令の書き方(単語)や組み合わせ方の規則(文法)
    - 使える命令や、組み合わせ方法は、言語によって異る

前期の復習：C 言語

- C 言語とは
  - プログラミング言語の一つ
    - プログラミング言語には色々あり得手不得手がある
  - 手続型: 命令を並べると、その命令がその順に実行される
    - 「手続(C 言語では関数)を定義する事」が「プログラミング」
  - コンパイラ型: C 言語で記述されたプログラムは翻訳される
    - C 言語で作られたプログラムは直接は実行できない(ソース)
    - C 言語から実行可能な形式への翻訳する(オブジェクト)
    - コンパイラ(ソースからオブジェクトへ翻訳するプログラム)が必要
    - オブジェクトコードはリンクされて実行形式になる
- MinGW (Minimalist GNU for Windows)
  - GNU Project の gcc (GNU C Compiler) の Windows 版
    - コンパイラは他にも色々ある (cf. Visual C++)
  - ソフトウェア概論ではこのコンパイラを使う
    - 実は C++ という C 言語を拡張した言語のコンパイラ

前期の復習：プログラム作成手順

- プログラム作成手順
  - 仕様: どんな機能にするか
    - プログラムの機能を考える
    - cf. 課題が出た
  - 設計: どうやって実現するか
    - 機能をどうやって実現するかを考える
    - cf. 課題を解く方法を考える
  - コード化: C 言語で記述する
    - プログラミング言語を利用して実現方法を記述
    - cf. C のプログラムを作成
  - デバッグ: 誤りがないか確認
    - プログラムが要求する機能をもっているか確認
    - cf. コンパイル/実行して機能を確認
  - 納入: 他の人に渡す
    - プログラムの使い方など資料を作る
    - cf. 課題を提出



(c) http://wwws.kobe-c.ac.jp/deguchi/c/step.html

前期の復習：プログラムファイルとツールの関係

□ プログラム作成の流れ

○ ソースファイル(*.c)

▹ C 言語で記述

▹ サクラエディタで作成

C> sakura hello.c

○ オブジェクトファイル(*.obj)

▹ ソースファイルからコンパイラで作成

C> cc -c hello.c

○ ライブラリファイル(*.lib)

▹ 予めコンパイラと一緒に配布される

▹ 自分で作ったり他から入手する場合もある

○ 実行ファイル(*.exe)

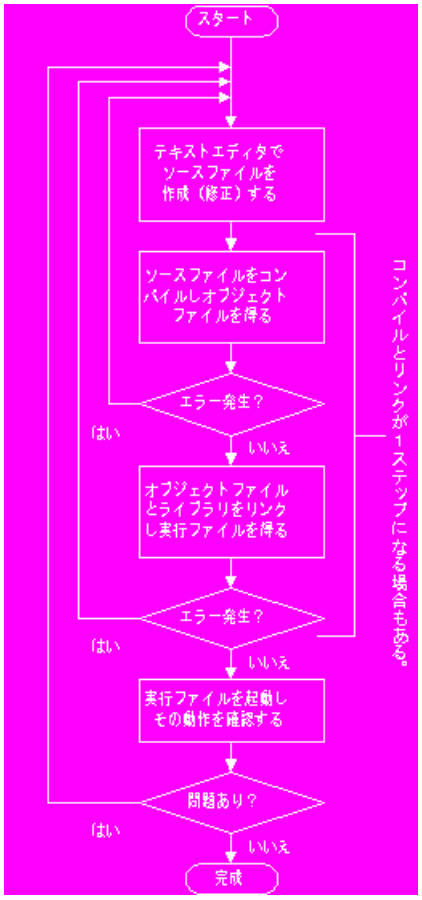
▹ オブジェクトファイルからリンカーで作成

C> cc hello.obj

○ 実行

▹ 実行ファイルを指定して実行する

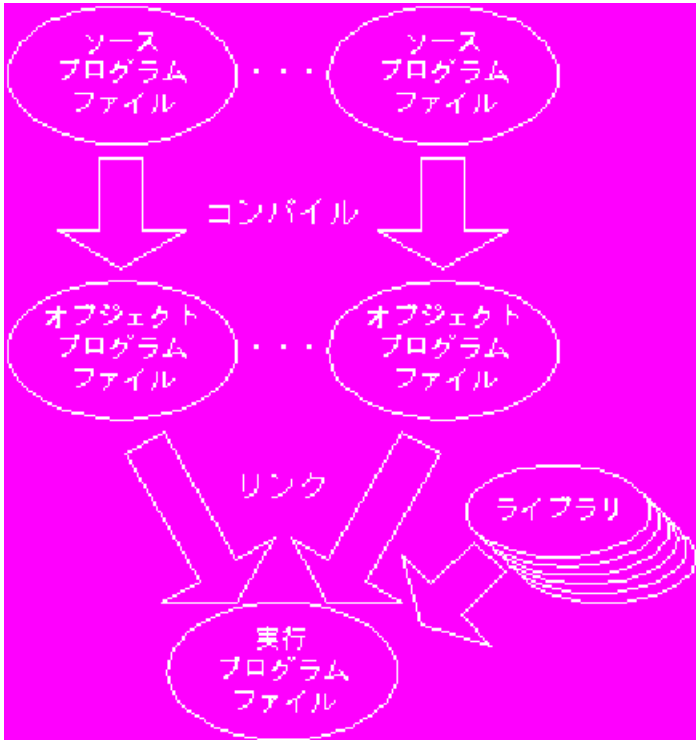
C> hello



(c) http://wwws.kobe-c.ac.jp/deguchi/c/step.html

前期の復習：様々なファイルの関係

- 様々なファイルの関係
  - ソースコード (*.c)
    - エディタで C プログラムを作成
  - オブジェクト(*.obj)
    - コンパイルがソースコードから作成
  - ライブラリ(*.lib)
    - コンパイラと一緒に配布される
  - 実行ファイル(*.exe)
    - リンカーがオブジェクトなどから作成



(c) http://wwws.kobe-c.ac.jp/deguchi/c/step.html

前期の復習：C 言語で Hello, World

- Hello, World プログラム (sample-001.c)
  - 「Hello, World[改行]」
  - 短いながら完全なプログラムで、意味がある
- プログラミング学習
  - 「動く」事が目標 (後期は「理解」も重視)
  - 差分プログラミング
    - 結果をすこしずつ作って行く
    - すでに動くプログラムの一部を変更する
- Hello, World の要素
  - main 関数
    - どのプログラムにも一つあり、最初に呼び出される
  - printf 関数
    - 「printf (引数文字列);」の形で呼出す
    - 引数文字列が画面に表示されるという「複作用」がある

前期の復習：C 言語の関数

- C 言語の関数
  - 定義: 関数は定義する事ができる
    - 「名前(型)」と「本体(手続)」をもち、それを結び付ける
    - 命令列を「{」と「}」でかこって、それに関数名をつける
    - cf. sample-001.c では「main 関数」が定義されている
  - 呼び出し: 関数は呼び出す事ができる
    - 「名前(引数並び)」で、関数を呼び出す事ができる
    - 呼び出すと本体に記述した命令列が実行される
    - cf. sample-001.c では「printf 関数」が呼び出されている
    - 関数呼び出しの結果として、関数値を得る事ができる
  - 関数呼び出しは、基本的な「命令」
    - 関数呼び出しによって、様々な命令が可能となる
    - cf.「printf 関数呼び出し」は「メッセージの表示」命令
  - 様々な標準関数 (ライブラリに定義されている)
    - printf - 文字列の出力機能を持つ
    - strlen - 文字列の長さを求める
    - strcmp - 文字列を比較する
    - 基本的な標準関数の名前と機能は、憶える(調べる知識)

前期の復習：関数の値と return 命令 (新)

- □関数宣言
  - ○関数の宣言の頭部の形式：「関数の値の型 関数名 (仮引数宣言)」
    - ▹「関数」は(実は..)値を返す事ができる
    - ▹関数が返す値(関数値)を「返り値(かえりち)」と呼ぶ
  - ○関数の値の型：返り値の型
    - ▹関数の値の型の「void 宣言」:「値を返さない」という特別な意味
    - ▹cf 仮引数宣言の時の「void 宣言」:「引数はない」という特別な意味
    - ▹[注意]「void：ボイド」:「空虚な、空っぽの」という意味
- □return 命令
  - ○「return 式;」で、「式」の値を関数の「返り値」として関数を終了する
  - ○[注意] return には二つの役目がある (というか、二つ同時に働いてしまう..)
    - ▹関数の終了：return 命令を実行すると、そこで関数の実行は終了する
    - ▹返り値の指定：その場合、return の後の式の値を「返す値」にする
  - ○[蘊蓄]
    - ▹return 命令がないと関数本体の最後(「}」の所)で、終了する
    - ▹void 型の関数でも「return ;」で、「値を指定せず終了」する事ができる

前期の復習：命令の組み合わせ

- 順接 (sample-002.c)
  - 命令を並べる
    - 並べた順に実行される
- 条件分岐 (sample-003.c)
  - if 文 (switch 文) で、条件と選択肢(複数の命令)を記述する
    - 条件によって複数の命令のどれか一つが実行される
- 再帰呼び出し (sample-004.c)
  - 関数の定義の中で自分自身を呼び出す
    - 関数の本体が何度も呼び出される
    - 同じ命令を何度も繰り返す事ができる
  - 再帰呼び出しの考え方
    - 「全部」を「扱いやすい一部分」と「残り(全部)」の二つに分ける
    - 扱いやすい一部分：これは、そのまま対処してしまう
    - 残り全部：(残り)「全部」なので、再帰呼び出しする

前期の復習：コーディング

- □ 計算機で何ができるか？
  - ○ 狭い意味：計算(情報処理)
    - ▹ 数の処理
  - ○ 広い意味：何でも(計算可能ものであればなんでも)
    - ▹ 原理的に現実でできることは何でも ??
  - ○ 「狭い」と「広い」の差は？
    - ▹ 「コーディング」で結び付けられる
- □ コーディングとは
  - ○ (扱いたい)対象を別の物(計算機の場合は数値)で表現する事
    - ▹ 座標：「位置」を「数値の組」で表現
    - ▹ MIDI：「音」を「音階を表す数値」と、「音の長さを表す数値」で表現
    - ▹ URL：「Web Page の位置」を「文字列 (http://～)」で表現
    - ▹ 文字列：「文字列」を「文字の並び」で表現
    - ▹ ASCII Code：「文字」を「数値」で表現
    - ▹ 2の補数表現：「数値」を「ビット列」で表現
  - ○ 「対象」を「数値」に対応つけるルール(コーディング)を考える
    - ▹ 「数値」を操作する事が、「対象」を操作する事になる：万能性

前期の復習：分割コンパイル

- プログラム
  - 関数の集まり
    - main とその他の関数定義を、ファイル内に記述する
- プログラムとファイルの関係
  - 一つのファイルに全ての関数を記述するか？
    - No : printf などは記述されていない
- 複数のファイルに跨がる関数をどう利用する
  - 分割コンパイルする (sample-005.c, sample-006.c)
    - cc -c でコンパイルし、obj ファイルを作成
    - cc でリンクし、exe ファイルを作成する
- プロトタイプ宣言と include
  - ファイルを分割すると関数の引数が解らない
    - プロトタインプ宣言で、それを報せる
    - include でプロトタインプ宣言を共有する

前期の復習：データ型

- 値には型がある
  - 様々な型
    - 文字型(char), 整数型(int), 浮動小数点型(double), ポインタ(*), etc..
  - 型によって、計算の方法が違う
    - cf. 3/2 は 1 に 3.0/2.0 は 1.5 になる
  - 型によって、表示の方法も違う
    - s_print_*****
- 引数(変数)は、値を保持する
  - 変数には、型宣言が必要
- 関数の返り値も、型宣言が必要
- 型変換
  - 値の間で意味を保持したまま型が変更できる
    - 例：1.0(double) <-> 1(int)
    - 同じ「1」という数値だが、型が変る
  - 型変換の方法
    - 明示的に行う(キャスト)：(int)1.0 -> 1, (double)1 -> 1.0
    - 暗黙に行われる：演算の時に「型の昇格」がおきる場合がある
    - cf. char -> int, int -> double

前期の復習：文字と文字型

- 文字とは？
  - シングルクォーテーション(「’」)で挟まれた一つの「文字」
    - cf. ’a’ は「a」という文字、’0’ は「0」という文字
    - 「\」(エンマーク/バックスラッシュ)はメタ文字(特別扱いの文字)
    - ’\n’ は改行をする文字、’\t’はタブ、’\a’ は音を鳴らす文字
    - ’\\’ は ’\’ 自身を表す、’\”で「’」を表現できる
- 文字の画面(標準出力)への出力
  - 「putchar (文字)」で、「文字」を出力できる
    - putchar (’a’) で「a」が出力される
    - putchar (’\a’) で音がなる

前期の復習：文字列

- □ 文字列とは？
  - ○ ダブルクォーテーション(「"」)で挟まれた「文字」の並び
    - ▹「\」(エンマーク/バックスラッシュ)はメタ文字(特別扱いの文字)
    - ▹「\n」は改行、「\t」はタブ、「\a」は音を鳴らす
    - ▹「\\」は「\」自身を表す、「\"」を使えば「"」を表現できる
- □ 文字列での「計算」
  - ○「+1」: 文字列に 1 を加える：先頭の文字が取り除かれる
    - ▹ "abc" + 1 は "bc" と同じ(ように振る舞う)
  - ○「頭に * を付ける」文字列の先頭の「文字」を取り出す
    - ▹ *"abc" は 'a' と同じ(ように振る舞う)
  - ○「後ろに「[数値]」」を付けると「数値番目の文字」が取り出せる
    - ▹「"abc"[2]」は「*("abc" + 2)」と同じで 'c' となる
  - ○ 文字列は「文字」の並びで、最後に '\0' (EOS : End Of String) がある
    - ▹ "abc" は 'a', 'b', 'c', '\0' の四つの「文字」が並んだもの
    - ▹ "" (空文字列) は、EOS 一つからなる

前期の復習：整数型

- 整数型
  - C 言語での整数
    - 表現できる範囲は限られている
    - 32bit の場合は −2147483648 から 2147483647
  - 宣言：int で行う
  - 計算：四則が可能 +, -, *, /
    - / は整数割り算なので、小数点以下は切捨てになる
  - 比較：大小比較、等号、不等号が使える
    - a > b : a が b より大きい
    - a >= b : a が b 以上
    - a == b : a と b が等しい (= でないことに注意 !!)
    - a != b : a と b が等しくない
- 整数型の出力 (当分は..)
  - s_print.h の中の s_print_int を使う (sample-001.c)
  - s_print_string で文字列が出力できる
  - s_print_newline で、改行

前期の復習：浮動小数点数型

- □ double 型
  - ○ 小数点付きの数を表現する
    - ▹ C 言語内での表現：小数点付きの数 (cf. 123.456)
    - ▹ 出力：s_print_double (s_print.h 内)
    - ▹ 入力：s_input_double (s_input.h 内)
  - ○ 様々な数学的な関数が利用できる
    - ▹ sin/cos/exp/log/etc.. cf #include <math.h>
  - ○ (当然) 四則の計算ができる：3.0/2.0 → 1.5 (cf. 3/2 → 1)
- □ 型の昇格と型変換
  - ○ 型の昇格
    - ▹ 浮動小数点数型と整数型が混在する計算では、自動的に浮動小数点数になる
  - ○ 型変換(キャスト)
    - ▹ 値の前に「(型名)」とすると、その型の数に変換される
    - ▹ 浮動小数点数を整数型にするには、キャストを利用する

前期の復習：代入

- 代入とは？
  - 変数の値を「変更」する事
    - 言い替えると..？「変数は代入が行われると変数の値が変化する」
  - 変数は常に値を持つが..
    - 代入された前と後では「値が変化」する
- 変数に値を代入するには？
  - 代入文 (「変数名 = 式」) を使う (sample-016)
    - 式の値が計算された上で、変数に代入される
    - 「=(等号)」を使っているが、「等しい」という意味ではない
    - C 言語では「等しい」という意味には「==」を使う
  - 変数の値は何度でも参照可能 (sample-017)
    - 計算結果を何度も利用する時には変数に入れておく
  - 代入の右の式の中で自分自身の現在の値が利用できる (sample-018)
    - やっぱり、「等号」じゃない

前期の復習：代入と関数引数

- 関数引数は「代入」か？(sample-006)
  - 代入は、「変更」がおきる
    - 元々変数がもっていた値が「失われて」しまう
  - 関数引数は、変数の値を決めるが、「変更」するわけではない
    - 変数の値が保存されている (sample-007)
    - 同じ「名前」を利用していても「異なる変数 (アドレスが異なる)」

前期の復習：局所変数宣言

- 引数でない変数
  - ブロックの先頭で変数宣言すれば、新しい変数が利用できる
    - メモリの一部がその変数名に割り当てられる
  - 引数との違い
    - 宣言直後は値が定まっていない(何が入っているかは不明)
    - 必ず、値を代入してから利用すること(可能なら初期代入すること)
    - 「初期代入」=,「変数宣言」+「代入」(実は微妙に違うが..)
- 名前の有効範囲
  - 「変数名」の有効範囲は宣言の後からブロックの終りまで
    - ブロックの外からはその名前が利用できない
    - 同じ名前で宣言しても異なるメモリになる(可能性が高い)
  - 名前とメモリは「独立」な事に注意
    - 名前は利用できなくてもメモリは利用できる(ポインターの利用)

前期の復習：入力・処理・出力

- 変数を利用したプログラムパターン
  - 入力・処理・出力の三つの部分からなる
    - 入力：変数を初期化する
    - 処理：(計算を利用しつつ..) 変数の値を変更する
    - 出力：変数の値を「結果」として出力する
  - 関数定義もこの形
    - 引数の値の設定、関数の本体、値の返却

コンピュータによる情報の表現

- コンピュータは電気で動く
  - 電気と「計算」の関係は？
- 電気回路のスイッチ
  - 「on：通電 / off：断線」の二つ状態を表現できる
    - cf. スイッチが on なら電球が光るが、off なら消える
  - スイッチの on/off を 1/0 に対応させる
    - 1/0 を「電気回路の on/off」でコーディングする
    - on と off, Yes と No, 真と偽, etc..
- スイッチ一つで表現できる「情報量」
  - 1/0 の二つの状態の内の一つが表現できる
    - 1 bit の情報が表現できる
    - bit はデジタルコンピュータが扱う情報量の最小単

bit 列による「情報」の表現

- 複数のスイッチ (bit) による表現力
  - n 個数のスイッチ (bit) で、2^n の状態が表現可能
    - 1 byte = 8 bit : 256 個の状態が表現可能
- コーディングによる情報の表現
  - コーディングの規則は「表」でよい
    - 「文字」の時は「ASCII Code 表」を使った
    - 半角英数記号の個数は 100 個程度なので 7 bit で表現可能
    - C 言語の「char 型」は 1 byte (= 8 bit > 7 bit)
    - 全角文字は 6879 文字 (JIS X 0208 の場合) あるので 1 byte では無理
  - 整数値 (-2,147,483,648～2,147,483,647) : 32 bit int
- sizeof 演算子
  - その型の byte 数を表す
    - sizeof(char) -> 1 / sizeof(int) -> 4

二進法

- 二進法とは
  - 2 を底とする位取り記数法
    - cf. 十進法は、10 を底とする位取り記数法
    - cf. 十六進法は、16 を底とする位取り記数法
- 二進法の性質
  - 数を 0/1 の二つの記号(数字)の並びで表現できる
    - cf. 十進法は、0 ～ 9 の十個の記号(数字)が必要となる
    - bit を単位とするコンピュータと相性がよい
- コンピュータでの数の表現
  - 基本は二進法の数 (0/1 の並び) と bit 並びを対応付けする
    - 非負の数値を表現する場合 : unsigned (負号なし) 数
    - 正負の数値を表現する場合 : signed (負号あり) 数

2の補数表現

- 1の補数表現とは
  - bit の 0/1 を反転させたもの
    - 例 (8 bit) : 01001000 -> 10110111
- 2の補数表現とは
  - 1の補数表現に 1 を加えた物
    - 例 (8 bit) : 01001000 -> 10111000
- コンピュータでの負号付き整数
  - 非負の数は、最上位 bit (負号 bit) が 0 で残りは二進数
    - 最上位 bit (負号 bit) で正負が判る
  - 負の数は、正の数の 2 の補数表現
    - 例 (8 bit) : 72 -> 01001000 / -72 -> 10111000

bit (論理)演算

- bit (論理)演算 (boolean)
  - 真偽値 (True / False) の演算
    - 論理積 (かつ / and / &) : 共に真なら真、それ以外は偽
    - 論理和 (または / or / |) : どちらか一方でも真なら真、それ以外は偽
    - 論理否定 (でない / not / ~) : 偽なら真、それ以外は偽
    - 排他論理和/論理差 (異なる / xor / ^) : 両方が異れば真、それ以外は偽
  - これら演算はスイッチで表現できる
- 1 bit の足し算 (ハーフアダー)
  - 1 桁の二進数の和は、2 桁になる可能性がある
    - それぞれの桁の計算は、論理演算で表現可能
- 論理演算による「計算」
  - 基本的な数値演算も論理演算の組み合わせで計算できる
    - 電気機械であるコンピュータが計算できる理由
    - より詳しい内容は、電子工学や電気工学の先生に聞こう

オーバーフロー

- コンピュータの情報
  - 有限のサイズをもつ
    - 表現可能な範囲が限られている
- 現実の世界 (例えば整数)
  - 無限の範囲をもつ
    - コンピュータの表現に対応しないものがある
    - 表現できる数を計算した結果が、再び表現できない可能性がある
- オーバーフローとは
  - 計算結果が表現可能な値の上限を超えること
    - signed char (8 bit : -128 ～ 127) の場合は 127 + 1 の結果は..
  - 浮動小数点数 (double) でも同様な事がおきる

型の性質

- コンピュータによる「情報」の表現
  - コンピュータでは数値(二進数)しか扱えない
    - 「情報」を「直接」は表現できない
    - コーディング(二進数と「情報」の対応)によって「間接」的に扱う
- 型
  - C 言語の中で「情報」の表現方法に名前を付けたもの
    - コーディングの名前と考えてもよい
    - 整数の表現 : int / 文字の表現 : char / 実数値の表現 : double
  - 型による違い
    - 計算結果が異なる : 3/2 -> 1, 3.0/2.0 -> 1.5
    - サイズが事なる : sizeof (int) -> 4, sizeof (double) -> 8

メモリモデルとポインター値

- メモリ
  - 複数のセルの並び
    - 個々のセルにはアドレス(番地)がつき、区別される
    - 個々のセルは 1 byte のサイズを持つ
    - セルの並びで一つの情報を記憶する (cf. sizeof)
- 変数とは？
  - 複数の連続したセルに「型」をつけたもの
    - 型によって、振舞が異なる事に注意
- ポインター値
    - 型と「アドレス値」をもつ
    - 「アドレス値」は、メモリの番地と同じ
    - 変数名の前に &' を付けるとポインター値が得られる
    - ポインター値の前に '*' を付けると元の変数と同じ振舞をする
      - 「 v == *(&v) 」が恒等的に成立する

ポインター値の計算

- ポインター値
  - 二つの情報をもつ
    - 型情報：何型の情報が入っているものか？
    - アドレス値：どこに入っているか？
- ポインター値の計算
  - 整数値 n を加える事ができる
    - 型情報は変らず、アドレス値だけが変化
    - アドレス値は n × sizeof (型) だけ変化 (n は負の数でもよい)
  - 同じ型のポインター同士なら引き算もできる
    - 結果は整数値で、(アドレス値の差) / sizeof (型) となる
    - p, q が同じポインター型なら「p + (q - p) == q」が恒等的に成立
  - キャストを利用して、型を変更できる
- ポインター値と添字
  - 恒等的に「 p[n] == *(p + n) 」が成立する

関数呼出しとメモリモデル

- 引数付きの関数呼出しの解釈
  - 「int func (x) { return x + 1; }」の時に、「func (5)」とは？
    - これまでは、「5 + 1」に置き換えて考えてきた (数学的解釈)
  - メモリモデルでの解釈
    - 「func (5)」: メモリのどこか (x という名前をつける) に 5 を保存する
    - 「return x + 1;」では、メモリ x から、5 を取り出して計算する
- C 言語ではどちらの解釈が適切か？
  - 実は.. メモリモデルになっている
    - では、数学解釈ではダメなのか : 実をいえば今迄の内容なら問題なし
  - この違いが問題になるのは？
    - 変数への「代入」操作が行われる場合