ソフトウェア概論 A/B

-- Hello World (三度) / ソフトウェア開発 --

数学科 栗野 俊一

2012/11/09 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

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

【復習】プログラム

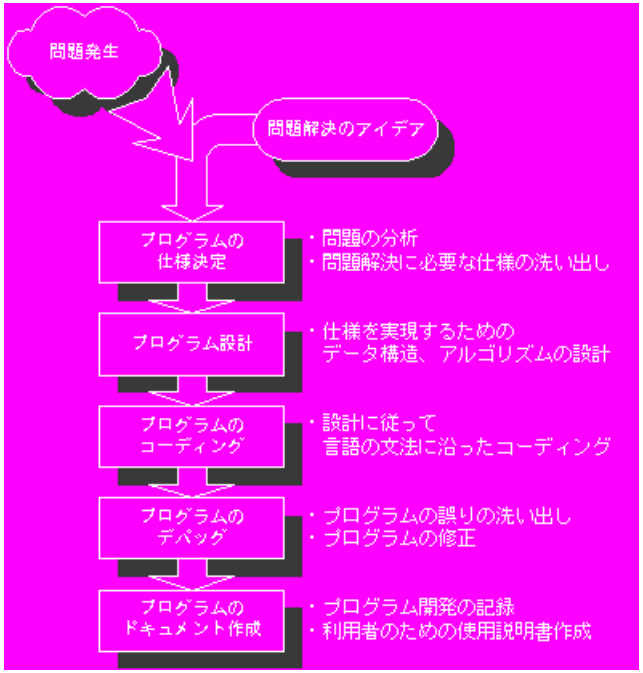
- □ プログラムとは
  - ○ 命令を並べた物
    - ▷ 命令:計算機に何か(単純な事..)をさせる事ができる
- □ 命令の組み合わせによって様々な機能(複雑な事)が実現できる
  - ▷「命令の組み合わせ方法」も考える必要がある
  - ○ 計算機はプログラムによって動いている
    - ▷ プログラムに記述された命令を順番に実施しているだけ
- □ プログラミングとは
  - ○ 命令を自分の意図に従って並べる
    - ▷ 計算機に自分の意図通りの動作をさせる事ができる
  - ○ 新しいプログラムを自分で作成する事ができる
- □ プログラミング言語とは
  - ○ プ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

【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

【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

【復習】プログラムファイルとツールの関係

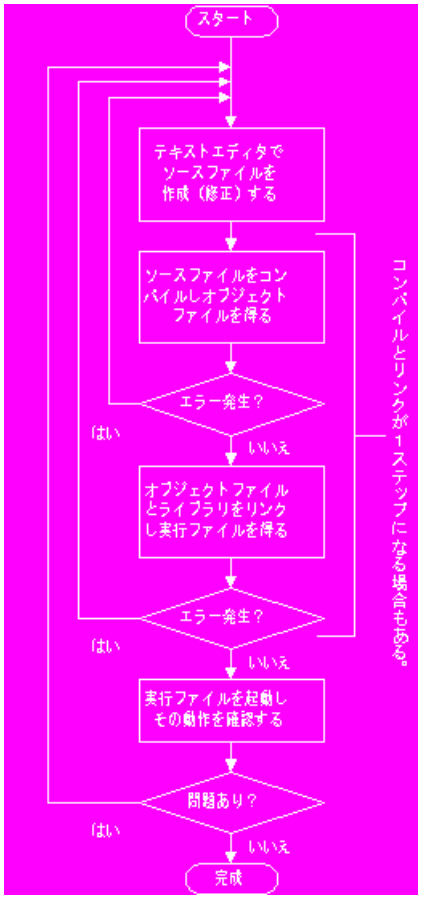
- □ プログラム作成の流れ
  - ○ ソースファイル(*.c)
    - ▷ C 言語で記述
    - ▷ サクラエディタで作成

    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

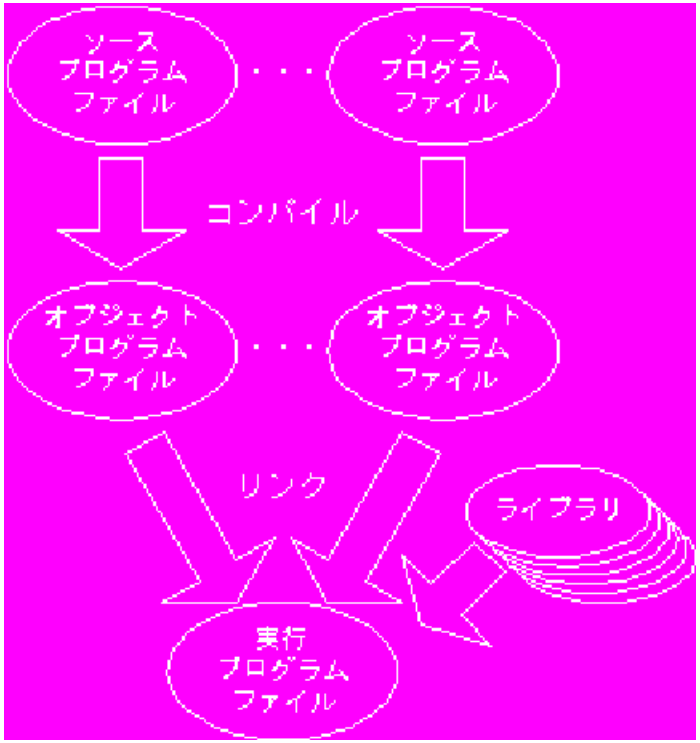
    C> hello



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

【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

【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

【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)

【復習】関数の値と return 命令

- □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

【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

【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

【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

【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

【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

【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

【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

【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

【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

【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)」

【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)

【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

【復習】メモリ

- □ (主)メモリ(記憶領域)とは？
  - ○ 情報を記憶する小さなメモリセルの集まり
    - ▹ 一つのセルでは 0 ～ 255 の 256 (= 2^8 : byte) 種類の状態の内の一つ(情報)が記録されている
    - ▹ 個々のセルには(その位置を表す)番地(アドレス)がついている
  - ○ [アナロジ] メモリ:ホテル, セル:部屋, アドレス:部屋番号, 情報:宿客
- □ メモリの操作：メモリセルの「記憶能力」(sample-001.c)
  - ○ 情報の記録 (set_memory_value_at)
    - ▹ アドレスと記録する情報を指定して、そのアドレスのセルに情報を記録する
  - ○ 情報の参照 (set_memory_value_at)
    - ▹ アドレスを指定して、そのアドレスのセルに記録された情報を取り出す
- □ メモリセルの性質
  - ○ 情報の参照は何度でもできる (sample-002.c)
    - ▹ 最後に記録した情報は何度でも取り出せる (cf. 不思議なポケット)
  - ○ 記録できるのは一つだけ
    - ▹ 最後に記録したものだけが記録され、参照できる (sample-002.c)
  - ○ メモリセルは独立 (sample-003.c)
    - ▹ 異るメモリセル(メモリセルの区別は番地で行う)は独立に振る舞う
  - ○ メモリセルの記憶容量 (sample-004.c)

【復習】メモリモデル

- メモリモデル
  - C 言語の変数のモデルの一つで、「変数をメモリセルの組み合わせ」として理解する
    - C 言語の「変数の振舞い」を「考えるための仕組み(モデル)」
    - !! 「何かモデル」とは何かを理解するために利用可能な、「より簡単な仕組み」の事」
    - !! 「C 言語の変数」を「メモリモデル」を通じて理解する/ 簡単なの理解しやすい
- 実は..
  - 多くの場合、「C 言語の変数」は実際に「メモリセルの組み合わせ」になっている
    - 変数の性質(代入)はメモリの性質(記憶能力)から説明できる
- char 型変数とメモリモデル (sample-005.c)
  - char 型変数は、一つのメモリセルだと考える事ができる
    - char 型変数は address を持つ
  - char 型変数をメモリセルと同様に扱う事ができる

【復習】メモリモデルと配列

- □ 文字列とメモリモデル (sample-006.c)
  - ○ 文字列は、文字の並び
    - ▷ 文字は char 型変数で記録できるので、文字列は char 型変数の並び
- □ 文字変数の並びと文字列 (sample-007.c)
  - ○ アドレスがわかれば、変数の内容をアドレス経由で操作できる
- □ 配列宣言 (sample-008.c)
  - ○ 配列とは
    - ▷「複数の変数の並び」の事 (個々の変数を「配列の要素」と呼ぶ)
  - ○ 一次元の配列宣言 (sample-008.c)
    - ▷ 変数と同様に型名の後ろに「配列名[サイズ]」の形で宣言
    - ▷「サイズ」の個数だけの変数が宣言される。
    - ▷ 配列の要素は「配列名[0] ～ 配列名[サイズ-1]」という「名前」になる
    - ▷ 添字：「[」と「]」の間には整数値が指定でき、配列の何番目の要素かを表す

【復習】文字列と文字型の一次元配列の関係

- 文字列と文字型の一次元配列の関係
  - C 言語の文字列
    - 文字の並んだもの (文字コードが連続に記録されている)
  - C 言語の文字型変数
    - 文字コードを一つだけ記憶できる
  - C 言語の文字型の一次元配列
    - 複数の文字型変数が並んだもの
  - C 言語の文字列型の一次元の配列で文字列を記憶する事ができる
- 文字列の一部の操作方法 (sample-009.c)
  - 文字配列の要素を変更すればよい
- 文字列を利用した文字配列の初期化 (sample-010.c)
  - 文字配列の要素を文字列を利用して初期化できる

【復習】配列の添字, 間接(参照)演算子,アドレス演算子

- 文字列の操作 (復習)
  - 「*」:間接(参照)演算子：文字列の先頭の文字を取り出す
    - *"abc" == a’
  - 「[]」:添字演算子：「[n]」で n (整数値)で「n+1番目の文字」意味する
    - "abc"[0] == ’a’, "abc"[1] == ’b’, ..
  - 「*」と「[]」の関係；文字列[n] == *(文字列+n)
    - "abc"[0] == *("abc"+0) == *("abc") == *"abc" == ’A’
- アドレス演算子「&」
  - アドレス演算子「&」は 間接演算子「*」の逆演算を行う
    - == "abc"
  - 変数に関しては逆が成立する
    - 「*(&var)」==「var」
  - アドレス演算子「&」の正体
    - 変数に対応したメモリセルの「アドレス」を得る演算子

【復習】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

【復習】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

【復習】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

【復習】多次元の配列

- 二次元の配列
  - 一次元の配列を二次元的に利用する事ができる
    - 二つの添字からアドレスを計算すればよい
  - 始めから二次元の配列を宣言する事ができる
    - 配列の要素のアドレス計算は、自動的に行われる
    - 実態は「(一次元)配列の(一次元)配列」だが、慣例により「二次元配列」と呼ぶ
  - 応用例
    - 行列は、二次元配列で表現可能
- 多次元の配列
  - 一つの型から新しい配列型を作るだけなのでいくらでも大丈夫(?)

【復習】整数型とメモリモデル

- 整数型とメモリモデル
  - 文字型変数はメモリセル一つに対応
    - では、整数型変数は ... ? / 実は、連続したメモリセルに保存される
- sizeof 演算子
  - その型の変数が、何個(byte)のセルをしめるかを教えてくれる
    - sizeof(char) == 1
    - sizeof(int) == 4 (32 bit の場合)
  - int 型の変数は 4 つのセルで表現される

【復習】ポインタ値とポインタ型

- ポインタ値
  - 「&」の作る値は実は、単なるアドレス値 *だけ* ではない
    - アドレス値も持つが、それと、「型情報」も持つ
    - 型情報：サイズ + 処理の仕方
    - !! 型情報は、コンパイル時だけ、実行時には解らない(解るのはアドレスだけ)
- ポインター値の型
  - 「型名 *」：「～型へのポインタ型」と読む
    - 「* をつけると「型」と同じになる」の意味
    - char * :「文字列」ではなく、「char 型へのポインタ型」だった

【復習】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

【復習】配列とポインタ型

- 配列とポインタ値
  - 「～型一次元の配列名」は、「～型へのポインタ型定数」となる
- 一次元の配列宣言とポインタ型変数宣言
  - 一次元の配列宣言「char a[N];」
    - char 型の変数 a[0] ～ a[N-1] の N 個の変数を宣言
    - 個々の要素変数の型は char 型
    - 配列名「a」は 要素の先頭を指すポインター型定数(a == &a[0])
  - ポインタ型変数宣言「 char *p; 」
    - 「char *」型の変数 p の 1 個の変数を宣言
    - 変数 p の値は不定 (だから *p の値も宣言時点では不定)
    - !! p の値は適切に初期化して利用する必要がある
  - 配列名によるポインター変数の初期化
    - 代入文「p = a」を行うと...
    - 「p[k]」と「a[k]」は全く、同じように振る舞う

【復習】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

【復習】データの混在した出力

- □ データの混在した出力
  - ○ 出力の場合に文字列の中に数値を含める事が多い
    - ▹ 一行の出力に複数の関数の呼び出し : 煩雑なかんじがする
  - ○ パターンが固定ならば、ある程度は簡便化できる
    - ▹ 単に関数にしても、お余り得した気分になっていない
- □ 文字列の中に数値を埋め込む事を考える
  - ○ 文字列の中に数値を表現する特殊な文字列を含め、数値に置き換える
    - ▹ '%' を特別な意味に利用
  - ○ 更に、複数の整数値が扱えるようにする
    - ▹ 引数が可変長になる (関数宣言の引数の "..." に注意)
  - ○ 整数値だが、8 進や 16 進で出したい場合も考える
    - ▹ '%' の後ろの一文字で判断 (d : 10 進 / o : 8 進 / x : 16 進)
    - ▹ '%' 自身を出力するために "%%" で、'%' を出力する
  - ○ 更に、色々な型の値の出力を考える
    - ▹ c : 文字 / s : 文字列 / f : 浮動小数点数
- □ printf : ライブラリ関数
  - ○ 書式付きの出力関数

【復習】printf の書式

- printf 関数
  - 色々な数値を出力形式(format)を指定して出力する関数
    - printf (char *format, ...); // 引数の個数は可変長
    - ex. printf ("答は %d です\n", 1 + 2 + 3); →「答は 6 です[改行]」
- 書式指定の一般形式
  - <書式指定> ::= (<文字> | "%%" | <書式>)*
    - 書式指定は、文字(それ自身)か、"%%" ('%' 一文字) か、書式(数値表示)
  - <書式> ::= '%' [<精度>] <型指定>
    - 書式は '%' で始まり、省略可能な精度記述の後に型指定がある
  - <精度> ::= ['-'] <整数> ['.' <整数>]
    - 精度は '-' (省略可能) を先行した整数で、'.' 以下が追加できる
  - <型指定> := 'd' | 'c' | 's' | 'f' | ..
    - 型指定は、一文字で、そのデータの型を表現する

【復習】scanf : 書式付の入力

- scanf 関数
  - 色々な数値を出力形式(format)を指定して入力する
    - scanf (char *format, ...); // 引数の個数は可変長
    - ex. scanf ("%d", &n); // n は整数型 / 引数はポインターを指定
- scanf の書式
  - 基本は、printf と同じ

本日の予定

- 講義
  - Hello World (三度目)
  - 設計
    - プログラムのパターン
    - Input-Process-Output
    - 解の探索
- 演習
  - 課題の提出

本日の課題 (2012/11/09)

- 今週 (2012/11/09) の課題
  - 課題 1:
    - ファイル名 : 20121109-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : printf/scanf の利用
    - 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)
  - 課題 2:
    - ファイル名 : 20121109-2-QQQQ.c (QQQQ は学生番号)
    - 内容 : 覆面算の「こな+ここ＝きなこ」を解く
    - 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)
- 先週(2012/10/26)の課題
  - なし

数学の存在証明とは

□ 数学の存在証明とは

$\exists x[P(x)]$: $P(x)$ を満す x がある事を証明せよ

[例]$\exists x[3x-6=0]$: 方程式 $3x-6=0$ の解はあるか？

○ 証明するためには、具体的な事例をみつければよい

$P(c) \rightarrow \exists x[P(x)]$: P を満す具体的な c があればよい

[例]$3 \times 2 - 6 = 0$: $x = 2$ とすれば方程式は満される

計算機の問題とは

□ 計算機の問題とは

$\exists x \in F[P(x)]$:「 有限集合」 F の中に $P(x)$ を満す x がある事を

[例]$\exists x \in \{$ 一桁の数 $\}[3x-6=0]$: 方程式 $3x-6=0$ の解は一桁の数

○ 有限集合の中から答を探せばよい

$$F = \{c_1, c_2, .., c_n\} : \text{有限集合}$$

$P(c_1) \vee P(c_2) \vee .. \vee P(c_n) \rightarrow \exists x[P(x)]$: $c_1, .., c_n$ の何れかが P を満

□ 計算機による探索による解法

○ すべての候補に対して、条件を満すかどうかを判定すればよい