ソフトウェア概論 A/B

-- 繰返し(再起呼出し) --

数学科 栗野 俊一 / 渡辺 俊一

伝言

私語は慎むように !!

- 色々なお知らせについて
  - 栗野の Web Page に注意する事

    http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino
- 講義開始前に済ませておく事
  - PC の電源を入れる
  - ネットワークに接続しておく事
  - 今日の資料に目を通しておく事
- 講義前の注意
  - 講義前は、栗野は準備で忙しいので TA を捕まえてください
- 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
- 本日の CST Portal の出席パスワード : 20140516
  - 出席は成績に影響しませんが、折角の機能なので、使いましょう

前回(2014/05/09)の復習

- 前回(2014/05/09)の内容
  - 再帰的関数定義
    - 関数を定義する時に、その本体にその定義を入れる関数が使える
    - 関数を再帰的(帰納的)に定義する仕組
    - cf. 階乗計算の！の定義：n! = n * (n-1)! と！の定義に！を使う
    - 関数は再帰的に定義されていると、(数学的に..)扱い易い
  - 文字と文字列
    - C 言語では「文字列("～")」と「文字('～')」は別物
    - 文字は、常に「一文字」を表現している
    - 文字列は、「文字の並び」を表現しているので 0 ～ n の文字がふくまれる

お知らせ

- 本日(2014/05/16)の予定
  - 作成：「繰返し」の習熟
  - 表現：ファイルの分割 / extern 宣言 / 文字比較
  - 操作：分割されたプログラムのコンパイルとリンク
- 本日(2014/05/16)の目標
  - 再起呼出しを利用した「繰返し」の習熟
  - 演習
    - 課題の提出

前回 (2014/05/09) の課題

- 前回 (2014/05/09) の課題
  - 課題 3:
    - ファイル名 : 20140502-3-QQQQ.c (QQQQ は学生番号)
    - 内容 : 底辺の長さが指定した文字列の二倍の長さ - 1 の横向のピラミッドを作成するプログラムを作成しなさい
    - 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)
    - 再帰呼び出しを利用する
    - 前回(2014/05/02)の課題の 3 なので、「20140502-3-QQQQ.c」である事に注意

本日 (2014/05/16) の課題

- 本日 (2014/05/16) の課題 (CST Portal のみ)
  - 課題 1:
    - ファイル名 : 20140516-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : 与えられた文字列を逆順に出力する rprintf を定義しなさい
    - 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 : 20140516-02-QQQQ.c (QQQQ は学生番号)
    - 内容 : 出力する繰返し回数を整数で指定する ntimeprint を作りなさい
    - 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 3:
    - ファイル名 : 20140516-03-QQQQ.c (QQQQ は学生番号)
    - 内容 : 負の整数も処理できる printint を作成しなさい
    - 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)

文字列と文字(再)

- C 言語での「文字」の扱い
  - 文字：文字をシングルクォート(「’」)ではさんだもの
    - cf. ’A’, ’a’, ’1’
    - 「一文字」と対応している「表現」
  - 文字の出力：putchar 関数を利用する
    - cf. putchar (’A’);
  - 文字列：文字列をダブルクォート(「"」)ではさんだもの
    - cf. "ABC", "123", "" (空文字列)
    - 「文字の並び(0個以上)」と対応している「表現」
    - <<注意>>：全角(日本語)は、一文字で、二文字分になる
  - 文字列の出力：printf 関数を利用する
    - cf. printf ("abc");
- 引数変数宣言における文字列と文字の区別
  - 文字の値を持つ変数の場合：「char」を使う
  - 文字列の値を持つ変数の場合：「char *」を使う
    - 「char * X」の意味は、「 X に * を付けて (*X) にすると char になる」

文字列の構造(再)

- 文字列は、文字の並び + 文字の終りからなる
  - "ABC" == { 'A', 'B', 'C', '\0' }
    - 長さ 3 (n) の文字列は、4 (n+1) つの部分からなる
    - '\0' を EOS (End Of String) と呼ぶ
  - k 番目の文字の取り出し方 : [k] をつける (k は 0 から始まる事に注意)
    - cf. "ABC"[0] == 'A', "ABC"[3] == '\0', "ABC"[9] == ? (未定義)
  - 先頭の文字を取り出すには * を付けてもよい
    - cf. *"ABC" == 'A'
  - 先頭の文字を取り除いた残りを取り出すには 1 を加えればよい
    - cf. "ABC" + 1 == "BC"
- 文字列の判定 : strcmp を使うと、「二つの文字列が同じなら偽」になる
  - cf. strcmp ("ABC", "ABC") ==> 偽, strcmp ("ABC", "XYZ") ==> 真
  - 文字列が「空文字列(長さが 0 / 文字を含まない文字列)」は先頭が EOS かどうかで判定できる
    - (*"" == '\0') ==> 真

「文字列」の再帰的な定義

- 文字列を「再帰的」に定義してみる
  - 文字列の再帰的な定義
    - "" (空文字)は文字列である ("" は '\0' だけからなる)
    - 先頭が文字で、残りが文字列なら、全体は文字列である
    - 上記の規則だけからなる物が文字列である
  - 参考(自然数の再帰的定義)
    - 0 は自然数である
    - n が自然数ならば n + 1 は自然数である
    - 上記の規則だけからなる物が文字列である
  - 比較
    - 文字列：先頭を取り除いた物 <-> 1 だけ引いた物：自然数
    - 文字列："" 空文字 <-> 0 (零)：自然数
- 自然数と文字列は似た構造(再帰構造)を持つ
  - 自然数と文字列の処理は同じにできる

再帰呼び出しの考え方(再)

- 目標
  - 「全部」をやりたい
    - でも一挙にはできない
- 対策
  - そこで問題を二つに分ける
    - 扱いやすい一部分：これは、そのまま、直接、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
- 注意点
  - 「全部」が「空っぽ」の時を忘れずに処理する
    - そうしないと「底が抜けて」しまう

ファイルの分割

- ファイルとプログラムの関係
  - プログラム：複数の関数からなる
  - ファイル：複数の関数を入れる事ができる
    - 一つのプログラム(の複数の関数)を一つのファイルに入れる事ができる
- ファイルの分割
  - 一つのファイルに一つの関数を入れてみる(理由は後述)
    - 一つのプログラムが複数の関数からなれば、複数のファイルが必要になる
- 分割コンパイル
  - 個々のファイルにある関数毎にコンパイルをする
    - ファイル毎に「cc -c foo.c」を使う (コンパイル)
    - 例 : cc -c foo.c → foo.o ができる
    - 例 : cc -c bar.c → bar.o ができる
  - コンパイルした複数のファイルをリンクして一つの実行ファイルにする
    - まとめて「cc -o」を使う (リンク)
    - 例 : cc -o foo.exe foo.o bar.o → foo.o, bar.o から foo.exe ができる
  - makefile の利用
    - 分割コンパイルする場合は makefile を作り make コマンドを利用すると簡単

ファイル分割の得失

- ファイル分割の得失
  - 得 (再利用 : 情報のコピー)
    - 同じ関数を使い回す事ができる
    - 一度作ったファイルは二度と作る必要がない(コンパイルも一度で済む)
    - 修正が必要になった時に困らない
  - 損
    - コンパイルの作業は面倒 : だから make がある
    - 外部宣言が必要となる
- 外部(extern)宣言
  - 同じファイル内に定義されていない関数を利用する場合に宣言が必要
    - 外部宣言「extern 関数頭部;」と、する必要がある
    - 例 : extern println(char *string);

ファイル分割と「コピペ」

- ファイル分割と「コピペ」は何が違う？
  - 機能的には、「コピペ」と変らないが..
    - 端的には、同じ物が沢山ある事が気に入らないが..
  - 「修正」が大変
    - 誤り(バグ:虫)が「コピペ」した部分に含まれていたら、全てを修正する必要がある
    - 「バグ」が「殖える」のは、「安易なコピペ」のせいだった..
  - 「甦るバグの恐怖：潰しても潰しても復活する..」は映画のタイトルでなく..
    - 「コピペ」した関数にバグがおり、一つだけ修正しても、他の部分が修正されていないので...
    - 「どこにコピペ」したのか解らない..
  - ゾンビバグ
    - 「全てたおしたはずが、エンデイングでまた、復活する」という恐怖は現実の物

繰返しのイデオム

- プログラムの作成の方針
  - イデオム(定型句)を身に付けると良い
    - このような事をしたい場合は、このような形でプログラムを作成する
- 繰返し(同じ事を何度もする)のイデオム
  - 再帰的関数定義を利用する
- 様々なイデオム
  - 逆順
    - 「先頭をしてから残り」か、「残りをしてから先頭」か
  - n 回数の繰返し
    - 長さ n の文字列を利用する
  - 回数とデータの分離
    - 回数の文字列と、対象文字列を分る
  - n x m の繰返し
    - 長さ n, m の文字列と再帰を二重に行う
  - 途中で特別な処理をする

整数(int 型)

- 整数(int 型)
  - 整数を表現する「型」
    - cf. char : 文字を表す型 / char * : 文字列を表す型(実は嘘だが..)
- 整数型の宣言
  - int を使う
- 整数型の操作
  - 四則(+,-,*,/)計算ができる
    - 余りの計算(%)もできる
- 整数型(文字型を含む)の比較
  - 等しい(==) / 異なる(!=) / 大なり(>) / 小なり(<) / 以上(>=) / 以下 (<=)
    - if 文の条件部分で利用可能