

プログラム言語論 抽象化

亀山幸義

筑波大学コンピュータサイエンス専攻

筑波大学 情報科学類 講義, 2010 年 5 月 31 日

抽象化 abstraction

抽象化とは?

- ということを**抽象的に** 論じるのはやめて、「型による抽象化」の**具体例**を見ることにしよう。

かな漢字変換ソフトウェア SKK

SKK の辞書ファイル:

おお **k** /大/多/尋/
きわ **m** /極/
たい **s** /対/大/對/
おこな **t** /行/
くら **b** /比/
...

辞書を操作する関数:

- 「見出し語」から「その見出し語を含む行」を得る。
- 「見出し語」に対する「変換結果」を加える。
- 「見出し語」に対する「変換結果」を削る。
- 「見出し語」を加える。
- 「見出し語」を削る。

SKK 辞書の変遷

- 開発体制: SKK 本体と辞書管理部分とは、別々の人が開発。
- なぜ、勝手に SKK 辞書フォーマットを変更してもうまく動いたか?
- SKK 辞書を使うための関数群の仕様を変更しなかったから。
 - 「見出し語」から「その見出し語を含む行」を得る。
 - 「見出し語」に対する「変換結果」を加える。
 - 「見出し語」に対する「変換結果」を削る。
 - 「見出し語」を加える。
 - 「見出し語」を削る。

フォーマットは変わっても、上記の 5 関数を使って得られる結果は常に同じ。

概要

前回まで

- プログラム言語の基礎: 構文と意味、インタープリタ、コンパイラ
- 束縛、ブロック構造とスタックに基づく評価、データ構造とヒープ、評価順序、制御構造、型システム

などの話題を、手続き型言語と関数型言語に基づいて議論した。
今回:

- データの抽象化・抽象データ型
- オブジェクト指向

Short Quiz

TWINS システムを再設計したい。

以下の設計のうち、どれを選択すると良いか、簡単な理由をつけて答えよ。

- 学群・学類ごとの module (情報学群の学生 module, ...)
- 処理の種類ごとの module (履修登録、成績登録、成績閲覧、GUI, ...)
- データの種類ごとの module (履修している授業データ、成績データ、学生の連絡先等の個人情報, ...)
- その他 (具体的に)

SKK 辞書のフォーマット

- 単純なテキストファイル、見出し語は順不同。
 - 辞書が大きくなってきたので、毎回ファイルの頭から 1 文字ずつ検索するのは遅い。
- 「行」単位で、見出し語の 50 音順でソート。
 - 「見出し語」自体が増えたり減ったりする。
- 「行」を 2 分木に格納。
 - 非常に巨大な辞書が作成され、もっと高速に検索したい。
- ハッシュを使って管理。
 - ... というのは浅薄。非常に巨大な辞書は (ハッシュする前の段階で既に) メモリには入りきらない。

この話のポイント

- 辞書操作の関数群を、使う人 (正確には、プログラムのうちそれらを使っているパート) と提供する人の合意事項が保たれれば、関数群の実装をどう変更しようと、使う人には影響がない。
- 辞書関数を使う人は、辞書が特定のフォーマットであることを使ってはいけない。(情報隠蔽, Information Hiding, カプセル化, Encapsulation)

- 今までのデータ型=具体データ型 (Concrete Data Type)
 - 新しく定義したいデータ型をどう構成したいかを、具体的に (型構成子を使って) 記述した。
 - そのデータ型が、具体的にどう実現されているかがわかって いる。
- 抽象データ型 (Abstract Data Type)
 - データ型の具体的な構成方法 (実現方法) は定めない。
 - データ型がどう使われるかだけを定める。
 - つまり、データの実装ではなく、データの仕様。

はじめに

型と抽象化

抽象データ型-3

stack の実装: 前ページの型と性質を満たす限り、どんな実装で もよい。

- **stack** を配列で実装。配列の第 0 要素が、スタックの底。
- **stack** を配列で実装。配列の最終要素が、スタックの底。
- **stack** をリストで実装。
- **stack** を配列で実装。ただし、メモリが不足すれば **malloc** 関 数でメモリを確保。

stack の利用: 前ページの関数を使う限り、どんな使用方法でも よい。

- 前ページの関数以外を使って、**stack** にアクセスしてはいけ ない。
- たとえば、**stack** の底のアドレスを得て、スタックの *n* 番目 の要素にアクセスする (**stack inspection**) のは禁止。

はじめに

型と抽象化

データ抽象化とモジュールの歴史

- CLU [1974-1975] by Barbara Liskov (2009 年 Turing 賞受賞)
- 抽象データ型/module 機能を使うことができる言語: ML, Ruby, Modula-2, Python, Perl, Fortran, COBOL, ...

はじめに

型と抽象化

ここまでのまとめ

モジュラリティ=大規模ソフトウェア作成における重要ポイント の 1 つ:

- 情報の隠蔽 or インターフェースと実装の分離。
- モジュール: モジュラープログラミングに対するプログラミ ング言語からのサポート (機能)。

stack: スタック (その要素は整数) をあらわす抽象データ型

- **stack** 型を操作する関数とその (具体) データ型。
 - **emptystack**: **stack**
 - **push**: **int*** **stack**→ **stack**
 - **pop**: **stack**→ **int*** **stack**
 - **isempty**: **stack**→ **bool**
- これらの関数が満たすべき性質。
 - **isempty(emptystack)**=**true**
 - **isempty(push(x,s))**=**false**
 - **pop(push(x,s))**=(**x**,**s**)

はじめに

型と抽象化

モジュール (module)

ソフトウェアの構成単位 (部品)
プログラムにおける、「何らかの関心事についてのまとまり」
インターフェースと実装から構成される。

- インタフェース (**interface**)
 - このモジュールを使うための仕様を定めたもの。
 - 通常は、モジュールを使うための関数の名前と型、など。
 - **stack** の場合、**push**,**pop**,**emptystack**,**isempty** 関数とその型。
- 実装 (**implementation**)
 - インタフェースが定められた関数等を実現するプログラム。
 - インタフェースに従う限り、どのような実装でもよい。
 - 実装のみに現れる関数は、外からは使えない。

はじめに

型と抽象化

モジュラリティ (modularity)

モジュラリティの高いプログラム

- 関心事ごとのまとまり (モジュールなど) が、それぞれ独立性 が高いこと。
- 独立性=インターフェースが実装と分離されていること。
- 「モジュラリティの高さ」についての定量的な尺度はない。

モジュラリティの高いプログラムの利点

- 各モジュールごとに独立に実装しやすい。
- プログラムの保守性・再利用性がよくなる。

はじめに

型と抽象化

Short Quiz (再掲)

TWINS システムを再設計したいとする。
以下の設計のうち、どれを選択すると **modularity** が**高い**か、簡単 な理由をつけて答えよ。

- 学群・学類ごとの **module** (情報学群の学生 **module**, ...)
- 処理の種類ごとの **module** (履修登録、成績登録、成績閲覧、 GUI, ...)
- データの種類ごとの **module** (履修している授業データ、成績 データ、学生の連絡先等の個人情報, ...)
- その他 (具体的に)