

プログラミング言語論

第3回

状態モデルと命令型言語(2)

データ型

担当: 犬塚

1

変数領域を確保する時期

- コンパイルの際 **static variable**
 - 実行時間効率がよい。型検査が容易。
- 実行時、変数の宣言に出会ったとき
stack-dynamic variable
 - 実行時ルーチンがスタック領域に変数を確保
 - 実行時でなければサイズなどが決定できない場合に有効。
 - メモリー効率を上げることができる。
- 実行時、その変数が必要になったとき
heap-dynamic variable
 - cのmalloc命令等によって領域を確保して用いる。
 - 領域は**ヒープ領域**から確保。不要な変数はfreeで解放。
 - 動的データ構造ではこの方法しかない。**無名変数。**

3

今日の講義

データ型に関する事柄を見る。

- 変数を確保する時期 静的／動的変数
- データ型
 - 基本データ型
 - ユーザ定義(構造)データ型
- データ型と集合の対応
- データ型と制御構造の対応
- 抽象データ型
- オブジェクト

2

基本データ型

primitive data types

- **数値型 Numeric type**
 - **整数型 integer type**
 - サイズによってさらに型を持つ言語が多い。
Java: byte, short, int, long.
 - **浮動小数点型 floating-point type**
 - 通常、精度によって型がある。float, double
 - **decimal (10進)型**
 - 通貨計算など精度のよい計算を行うための型。COBOL, C#。
- **論理型 Boolean type**
 - 真偽を表す型。ALGOL60, C等では特にこの型を持たず、数値型で真偽を表す。
- **文字型 Character type**
 - 8bitのASCIIコード文字を表す→16 bit unicode。

4

文字列型

- 文字列型の扱いは、言語によっていろいろ。
- 主に次の2つの方針
 - 文字型の配列として文字列を扱う。
 - C, C++, PASCAL
 - この場合は文字列を扱うための演算(接続、代入など)自分で用意する。(PASCALのように、一部用意されている場合もある)
 - 文字列を扱うための特別な型を用意。
 - JAVA, C#, LISP
 - この場合は、文字列演算や関数が用意されている。
- 文字列の長さについての方針
 - 長さを最初に決めてその後は変えられない。 static
 - 予め宣言した長さまで、いくらでも変えられる。 limited-dynamic
 - 後からいくらでも長さを変えられる。 dynamic 5

ユーザ定義型（構造データ型）

- 列挙型
- 部分範囲型
- 配列型
- レコード(構造体)型
- ユニオン(共有体)型
- 参照(ポインタ)型
- その他

6

列挙型、部分範囲型

enumeration types, subrange types

□ 列挙型=任意の名称の定数の有限集合の型

type days=(Sun, Mon, Tue, Wed, Thu, Fri, Sat)

- 列挙型は順序型の1つ。
- 順序型は離散的かつ順序のある型: 整数、論理、文字。
浮動小数点数は離散的でないので順序型でない。

□ 部分範囲型=順序型において、その範囲の一部を指定する型

type wdays=Mon..Fri;
months=1..12;

配列型 array type (1)

- 同一のデータ型要素(要素型, element type)の集合体。
添え字(subscripts, index)で各要素を指定。
- A[i] — 添え字の型(添え字型, subscript type)と
要素の型(要素型, element type)で決まる。
- 添え字の指定でその変数領域を定数時間で参照できる。
- 配列変数は、添え字の範囲→要素型への写像とみなせる。
(有限写像)

配列は次のバリエーションがある:

- 配列の大きさがコンパイル時に決めるか、実行時にきめるか。
- 記憶領域の割当てをコンパイル時に行うか、実行時に行うか。

7

8

配列型 array type (2)

正確には、3通りある。

- コンパイル時
 - 実行中の宣言の解釈時
 - プログラムの要求中。
- | | |
|-----------------------------|---------------------|
| □ static array | コンパイル中に大きさ、領域とも固定。 |
| □ fixed stack-dynamic array | 大きさは固定。領域は宣言解釈時。 |
| □ stack-dynamic array | 大きさ、領域確保とも宣言の解釈時。 |
| □ fixed heap-dynamic array | 大きさは固定。領域はプログラムで要求。 |
| □ heap-dynamic array | 大きさは領域ともはプログラムで要求。 |

stack-dynamic → i=10; int a[i]

heap-dynamic → malloc をつかった確保。→freeで領域を解放。

9

配列に関する検討事項

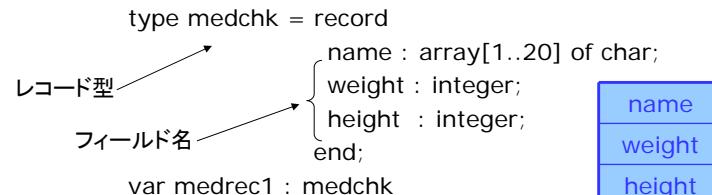
□ 配列演算

- 配列をまるごと演算できる言語もある。APL
- 配列をまるごと代入、足し算など。

レコード(構造体)型 record type

□ 異なったデータ型を1つにまとめるデータ型。

- record: COBOL, pascal,... struct: C, C++.



□ フィールドの参照。

- フィールド of レコード型変数 height of medrec1
- レコード型変数.フィールド medrec1.height

□ レコード型の演算 代入

11

ユニオン(共有体)型 union type

□ 1つの変数に異なったデータ型の値を記憶できる型。

```
union {
    int ival;
    float fval;
    char cval;
} a;
```

変数 a は int, float, char のいずれかの型の値をとる。
参照: a.intval = 3

ivalまたはfvalまたはcval

□ どの型が実行時に入るかわからないので、型検査はしない。

□ discriminant union = 型検査あり。ALGOL,PASCAL,Ada

```
type status = (married, single)
paeson = record
    name : array[1..20] of char;
    case st : status of
        married : (mdate : date);
        single : (age : integer);
    end
```

name
st

st=marriedの場合:mdate
st=singleの場合:age

参照(ポインタ)型

reference type, pointer type

ポインタ = 指し示す変数のデータ型

+ 変数のアドレス情報または特別の場合の値(nil)

2つの働きがある:

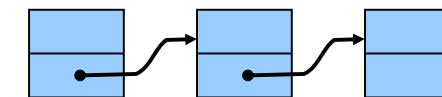
- 間接的にデータを指し示す(間接アドレッシング)
- ヒープ領域に変数を確保したときのような無名変数(anonymous variable)を扱う。

- ポインタの型演算子: * (c)、^ (PASCAL)、access (Ada)
- ポインタは他のユーザ定義型のように構造を作らない。
- ポインタ型変数は値を保持するというより、値を保持する記憶領域の場所を保持する = 参照型 reference type

13

ポインタ型と動的データ構造

- ポインタ型は動的データ構造生成のために用いられる。 linked list, tree, ...
- heap-dynamic variableとして動的に確保した無名変数を保持するために、ポインタを用いる。



ヒープ領域から確保した無名変数

- ポインタの代入の際、注意が必要: 領域を解放してポインタだけ残った行先無しのポインタ dangling pointer になる危険がある。

14

その他のデータ型

- 連想配列
- 集合型
- リストなどの特定用途の構造

- その他の特別な型
 - ファイル型、入出力ストリーム型
 - 通信ポートなど

15

データ型と集合演算

- データ型は変数の値の範囲(変域; range)を決めるもの。
- データ型とその型の変域である集合が対応。
integer ⇔ 整数の集合 等

- ユーザ定義の構造型は、集合演算に対応する。
 - 列挙型 ⇔ 有限集合の定義
 - 部分範囲型 ⇔ 部分集合
 - レコード型 ⇔ 直積
 - ユニオン型 ⇔ 和集合
 - discriminant union ⇔ 直和
- 集合型 ⇔ べき集合
- 配列 ⇔ 添え字型 → 要素型の関数の集合

16

データ型と制御構造

- さらに、制御構造ともある程度の対応がある。

レコード型 \Leftrightarrow 逐次実行(合成)

ユニオン型 \Leftrightarrow 非決定的分岐

discriminant union \Leftrightarrow 条件分岐

配列 \Leftrightarrow ループ(一定回数の)

動的配列 \Leftrightarrow ループ(不定回数の)

linked list \Leftrightarrow 再帰(末尾再帰)

- 類推に過ぎないところがあるが、その型の変数の変域と、実行の実例の集合をある程度対応付けられる。

17

型検査 type check

つぎの場合、**型の違反 type error** となる。

- 異なった型の間の代入、
- 演算において許されない型の変数への適用、
- ユーザ定義の手続きにおいて、定義と異なる型の変数の受け渡し、

型検査のタイミング

- コンパイル時: **静的型検査 static type checking**
- 実行時: **動的型検査 dynamic type checking**

18

強く型付けられた言語

strongly typed languages

- 構造的プログラミングとともに提唱。
- 各変数、関数(その引数と返り値)等のすべてがコンパイル時に型をつけられている言語。
- もし型の違反があれば、かならず型検査で察知される。
- したがって、discriminant ではない共有体をもつ言語(c,FORTRAN等)は強く型付けられた言語でない。

19

型の整合性 type compatibility

- 型が一致する場合、その型の変数の間で代入を行うことができる。
- しかし、次の場合に、代入を禁止してよい場合は、検討の余地がある。多くの言語では代入を許す。
 - 整数型の変数と、整数の部分範囲型の間。
 - 同じ型のフィールドをもつ別の型の構造体同士。
たとえば、複素数と2次元座標。
 - integer A[0..10]の配列と、integer B[5..15]の配列。

20

型の自動変換 type conversion

- 型の合わない変数間で代入、演算をする場合に自動変換をすることがある。
 - $1 + 2.0$ int型をfloatに自動変換して演算
 - $X := 1$ Xがreal型でも、1を自動変換する
- こうした自動変換は、**強制変換 coercion**という。
- 強制変換は便利であるが、型検査と逆行する。

21

☆レポート課題

- 代表的な命令型言語パラダイムのプログラミング言語について、データ型を調査してください。
- 2つ以上の言語で調査して比べて見なさい。
- 命令型言語
 - FORTRAN、COBOL、C、PASCAL、BASIC、Perl、APL、Ada、Modula-2
 - Javaはオブジェクト指向言語ですので、今回は含めない。
- 基本型の種類
- ユーザ定義型の種類、その書き方
- その他の特徴、など

5月10日の講義時提出

22