

計算機言語講義資料 第3回 命令型プログラミング言語(1)

大阪大学 大学院情報科学研究科

2015年

長谷川 亨

t-hasegawa@ist.osaka-u.ac.jp

アウトライン

- 文
 - 構造化プログラミング
- データ型
 - 基本データ型と構造型
 - データ型の実現
 - ポインタ型と右辺値、左辺値
- 手続き
 - 引数結合
 - 名前の有効範囲と存続範囲
 - 再帰呼び出しと実行時スタック
- その他の話題
 - 再帰呼び出しの無い形への変換、副作用など

命令型言語

- 計算モデル
 - フォンノイマン型コンピュータ (第1回講義 理論モデルRAMを参照)
- 基本要素
 - 変数名
 - 文...代入文、制御文
- 特徴
 - 代入文の繰り返しで計算
 - プログラムは文の並び(列)であり、逐次的に実行
 - 代入文の実行で、変数名の値を動的に変化させ計算

2.2節

文

文

命令型言語のプログラム

- プログラム=宣言部+実行部
- 文は計算の状態を変化させる
- 文のBNF
 - $\langle \text{文} \rangle ::= \langle \text{代入文} \rangle | \langle \text{制御文} \rangle | \langle \text{名前} \rangle : \langle \text{文} \rangle |$
 $\text{begin } \langle \text{文} \rangle * \text{end}$
 - $\langle \text{制御文} \rangle ::= \langle \text{条件文} \rangle | \langle \text{繰り返し文} \rangle | \langle \text{ジャンプ文} \rangle$

代入文

代入文

変数 := 式

の形式をしている (C言語の場合、変数 = 式)

- ① 右辺の式の値を求める(右辺を評価する)
- ② 左辺の変数に値を代入する

C言語の例

```
sum=sum+sq;
```

(C言語では代入文と呼ばず、代入演算子と呼ぶ)

テキストには記載
されていません

6

構造化プログラミング

<無条件ジャンプ文> ::= goto <名前>

C言語

- goto文
- 「プログラミング言語C」3.8 Gotoと名札
 - 「ここにあげたような少数の例外を除くと、goto文に頼るプログラムは、goto文を使わないプログラムに比べて、一般によりわかりにくく、また保守しにくい。この問題についてわれわれは別に教条主義者ではないが、goto文はとにかく滅多なことでは使うべきでないと思う。」

構造化プログラミング

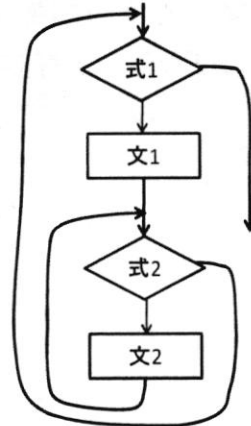
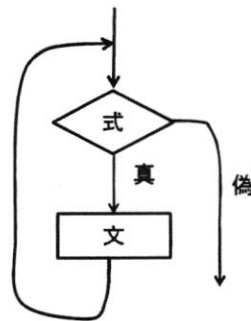
構造化プログラミング

- プログラム中の全てのループが繰り替えし文のみにより作られる時、プログラムは構造化 (well-structured) されているという
- ループの入口と出口
 - すなわち、各ループの入口と出口が一つであり、ループが入れ子構造になることが保証される
- 入れ子構造
 - ループが入れ子構造になることが保証される

構造化プログラミング

```
while(<式>){
  <文>}

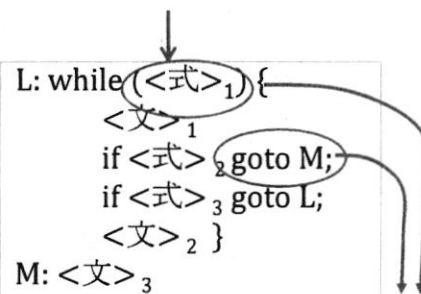
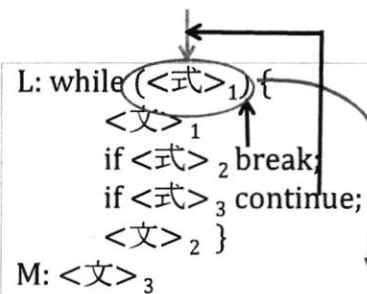
```



構造化プログラミング

C言語のbreak文とcontinue文は、この良い構造を壊さない

- break文 繰り返し文を終了して、繰り返しのつぎの文を実行する
- continue文 繰り返し文の先頭に制御を移す
- break文/continue文とgoto文の使用例



構造化プログラミングへの批判

goto文を使用しない、構造化プログラミングは有用であるが、常にそうとは限らないとの批判もある

「プログラミング言語C」3.8 gotoと名札の、P80～P81に、上記の「そうとは限らない」例が記載されている。

テキストには記載
されていません

11

演習

【演習問題6】 goto文を使用した方が簡潔なプログラムを書ける場合を示せ。

12

データ型

13

データ型

- 実行効率の良いアルゴリズムの設計には、適切なデータ構造を選ぶ(または創造する)ことが重要
- プログラミング言語は、複雑なデータ構造を容易に提供する
- 型付けされた言語
 - データを格納するために用いる名前(変数名)を使用する前に、データ型を宣言することを必要とするプログラミング言語
- 基本データ型(組み込み型)と構造型
 - 基本データ型
 - 整数や実数などの基本的なデータ構造を表現
 - 構造型
 - 基本データ型などの構成要素持つ、複雑なデータ構造を表現
 - 配列型, レコード型など

14

基本データ型

文字、整数、実数、論理(真, 偽)などの基本的なデータ

整数型, 実数型, 論理型

単一の値を持つ

基本データ型はハードウェア(計算機)の特性を反映している

– ハードウェアの特性を反映しているが, その具体的な表現はプログラマに見えない

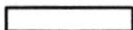
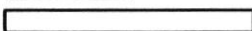


- 例)整数はレジスタ長(例えば, 32ビットマシンの場合は32ビット長)に対応して, 32ビット固定小数点でハードウェアでは実現されることは, プログラマに見えない
- 利点
 - 安全なプログラミング: ハードウェアでの表現に依存したプログラムを避ける
 - ポータビリティ(移植性): 抽象的な演算のみを用意しており, 表現の細部は利用しないので, 表現の異なる計算機に移植可能

35

基本データ型の実現

ハードウェアを見せないといいつつも, 影響を色濃く受ける

基本データ型の実現例

| | | |
|-----|---|------------------------------|
| 整数 |  | 16ビット固定小数点 |
| 実数 |  | 32ビット浮動小数点 |
| 論理型 |  | 1バイトのデータ. 偽/真はビットパターンがオール0/1 |
| 文字型 |  | 1バイトのデータ(バイト単位で番地つけできる場合) |

36

演習

【演習問題7】 16 ビット計算機用の C言語 で書いたプログラムが 32 ビット用の C言語でコンパイルして動いている. 32 ビット用の C言語で書いたプログラムを 16 ビットマシンにもってきて, コンパイルして動くだろうか? どういうことがありうるかを検討せよ。

17

C言語での配列の実現

一次元配列の実現

- 例 `int a[100];`
- 配列aの要素を $a[0], a[1], \dots, a[99]$ の順にコンピュータのメモリ領域に割り当て
- $a[i]$ に対応する(相対)番地は $\text{base}_a + i * w$ で計算される ($0 \leq i \leq 99$)
 - base_a は $a[0]$ の最初の番地
 - w は要素型(ここではint)の要素を格納するのに必要なサイズ(バイト数)

18

C言語での配列の実現

二次元配列の実現

- 例 `int b[2][3];`
- 配列bの要素を`b[0][0]`, `b[0][1]`, `b[0][2]`, `b[1][0]`, `b[1][1]`, `b[1][2]`の順にコンピュータのメモリ領域に割り当て
- `b[i][j]`に対応する(相対)番地は $\text{base}_b + i*3*w + j*w$ で計算される ($0 \leq i \leq 1, 0 \leq j \leq 2$)
 - base_b は`b[0][0]`の最初の番地
 - w は要素型(ここでは`int`)の要素を格納するのに必要なサイズ(バイト数)

このような配置を、行優先配置と呼ぶ

19

演習

【演習問題8】

C言語のプログラム中で、`int b[2][3][4];`と宣言されているとき、配列要素はどのような順番でメモリ領域に割り当てられるかを答えよ。

20

ポインタ型

プログラム中で宣言された変数は、コンピュータのメモリ領域に、変数を保持するための領域(メモリセル)が割り当てられる

- 左辺値と右辺値
 - 変数 x が a_x 番地の領域に割り当てられると、 a_x をアドレスあるいは左辺値という、
 - 一方、 x の現在の値を右辺値という
- ポインタ(ポインタ変数)
 - その値(右辺値)として、ある変数のアドレス(左辺値)を持つもの

左辺値と右辺値

- 代入文を用いた説明 代入文 $x = y;$
 - 変数とは値を記憶するための「場所」で、右辺の式が評価されてその「値」が変数に格納される
 - 左辺は、値を記憶する場所を指示する: 左辺値
 - 右辺は、代入する値を示す: 右辺値
- 単項演算子*と&
 - 変数及び配列要素は、左辺値を持つ
 - 左辺値を取り出す単項演算子&
 - x が変数のとき、 $\&x$ の値は x の左辺値である
 - 逆の演算子* 間接演算子と呼ばれる

左辺値と右辺値

- `int x; int *y;`
 - ポインタ変数`y`は、`&x`を値として持つことができる
 - `*y`が`int`型の変数と同様に用いることができることを示している
- 記法
 - $l(e)$: 式 e の左辺値、 $r(e)$: 式 e の右辺値、 \perp : 未定義
 - 単項演算子`*`
 - 式 e が右辺値を持つとき $l(*e)=r(e)$
 - 単項演算子`&`
 - 式 e が左辺値を持つとき $r(&e)=l(e)$

25

左辺値と右辺値

- 例4.8
 - `int *y;`のとき
 - $l(*(&y))=r(&y)=l(y)$ ($=y$ のアドレス)
 - $r(*(&y))=l(*y)=r(y)$
 - $r(&(*y))=l(*y)=r(y)$
 - $l(&(*y))$ は未定義

26

その他の話題
テキストには記載されていません

型変換

タイプ変換(型変換)

C言語やPascalでは型が異なると、代入や演算は許されないのが普通である

しかし、実数型の変数に整数型の式を代入する代入文を書いても良い

例えばPascalではできるだけ型を厳密に扱うという考えで作ったが、このような例外がある。プログラマにとって便利であるからと考えられる。

テキストには記載
されていません

型変換

(1)暗黙の型変換 コンパイラが自動的に型変換を行う

(1-1)代入時の変換

右辺と左辺の型が違う場合に、左辺の型に変換する

```
int a; double x=3.1415; sa =x;
```

aは小数点以下が切り捨てられ、3になってしまう

(1-2)式の中で行われる変換

式中で異なる型の定義や変数が現れるとき、一番精度の高い型に統一する

```
double d; long l; int i;
```

```
if (d>i) d=i;
```

```
if (i>l) l=i;
```

```
if (d==l) d*=2;
```

テキストには記載
されていません

型変換

(2)明示的な型変換(キャスト)

強制的に別の型に変換する

```
int a=3; int b=2; float f;
```

```
f=(float)a/b;
```

【演習8】 $f = a/b$; と $f = (\text{float})a/b$; を実行後の、それぞれのfの値を示せ。

テキストには記載
されていません

結合固定、バインディング、束縛

バインディング 束縛

プログラムやコマンドで指定された事項がいつ決まるかが問題である

決定時をバインディングタイムと呼ぶ

配列に関する指定がいつ固定されるか

コンパイル時 (バインディングタイムがコンパイル時)

配列の次元, 要素型, 添字型が決まる 静的配列

実行時に要素型, 添字型を変更できない

オブジェクト生成時

その型の変数を生成する時に添字型が決められる

実行時に必要な大きさの配列を作れるが, オーバヘッドが大きい

オブジェクト操作時

添字の範囲がそのオブジェクトの生存時間内において, 可変だが,

オーバーヘッドが大きい

インタプリティブな言語APLなどで提供

テキストには記載
されていません

第3回回答

【演習問題6】

```
for (i=0; i<n; i++)  
    for (j=0; j<m; j++)  
        if (a[i]==b[j])  
            goto found;
```

found:

【演習問題7】

正しく動作しない可能性が高い。

intは32ビット用のC言語では32ビット、16ビット用のC言語では16ビットである。例えば、int型の変数に、 2^{16} 以上の値は代入できない。

【演習問題8】

b[0][0][0], b[0][0][1], b[0][0][2], b[0][0][3],
b[0][1][0], b[0][1][1], b[0][1][2], b[0][1][3],
b[0][2][0], b[0][2][1], b[0][2][2], b[0][2][3],
b[1][0][0], b[1][0][1], b[1][0][2], b[1][0][3],
b[1][1][0], b[1][1][1], b[1][1][2], b[1][1][3],
b[1][2][0], b[1][2][1], b[1][2][2], b[1][2][3]

【演習問題9】

| | |
|-----------------|-----|
| f = a/b; | 1 |
| f = (float)a/b; | 1.5 |

【演習問題9】

| | |
|-----------------|-----|
| f = a/b; | 1 |
| f = (float)a/b; | 1.5 |