計算機言語講義資料 第4回
命令型プログラミング言語(2)

大阪大学　大学院情報科学研究科
2014年
長谷川亨
t-hasegawa@ist.osaka-u.ac.jp

手続き

- 手続き
  - 部品化されたプログラム単位
    - 大規模で複雑な処理を行うプログラムを設計するには、要求された機能をいくつかのより簡単化された機能に分解し、これらの機能を組み合わせることによって元の機能を実現する部品化(モジュール化)が重要
    - アセンブリ言語なので、まとまった作業を行う部分を切り出したサブルーチン，副プログラムも同様なもの
    - 手続きの中で計算結果を返すものを関数という

手続きのBNF

手続きのBNF

<手続き定義>::=
<返り値の型名>
<手続き名>
{<仮引数>$_1$, <仮引数>$_2$, ・・・, <仮引数>$_n$}
<手続き本体>

<手続き本体>::=<複文>

<手続き呼び出し>::= <手続き名>{<実引数>$_1$,
<実引数>$_2$, ・・・, <実引数>$_n$}

3

サブルーチン型手続きと関数型手続き

- サブルーチン型手続き
  - 計算結果の情報は、パラメータ(引数)のどれかを用いて、あるいは手続きとそれを呼んだプログラムの共有する変数(グローバル変数)を介して返す
  - call 手続き名(実引数のならび)
- 関数型手続き
  - 計算結果の情報は，その手続き名が現れるところにその値が返される
  - 式の中に呼び出しを書く
  - 例 dist := sqrt(sqr(x1-x2)+sqr(y1-y2));
  - 関数sqrt(), sqr()

テキストには記載されていません

4

実行時スタック

手続き、関数の呼び出し毎に、

仮引数、計算の途中までの結果やプログラムカウンタの値などの制御情報を保存する必要がある

呼び出し毎に必要な情報を格納するためのメモリ領域を駆動レコード(activation record)

実行時スタック

実行時のレコードを格納するために，通常スタックを用いる

スタック

後入れ先出しのデータ構造

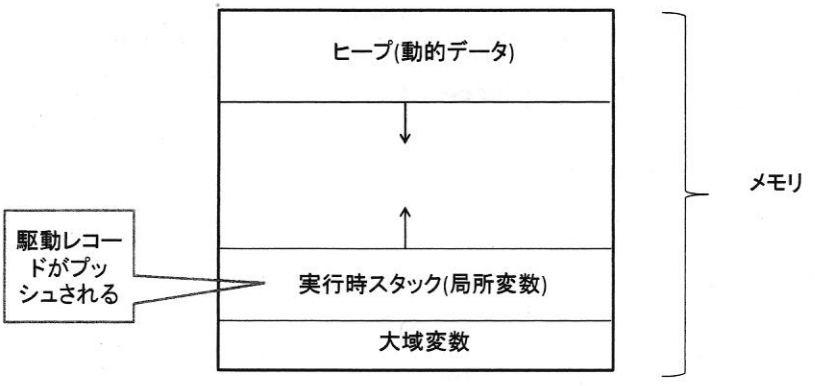
データ構造のプッシュとポップ

(注)テキス図4.7「実行時のスタックの変化」参照

Cプログラムの変数割り当て

ヒープ　動的データを格納

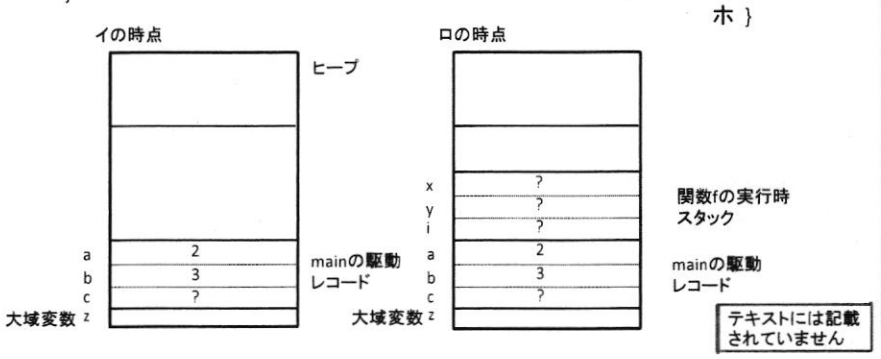
スタック　局所変数や仮引数などの局所変数を格納

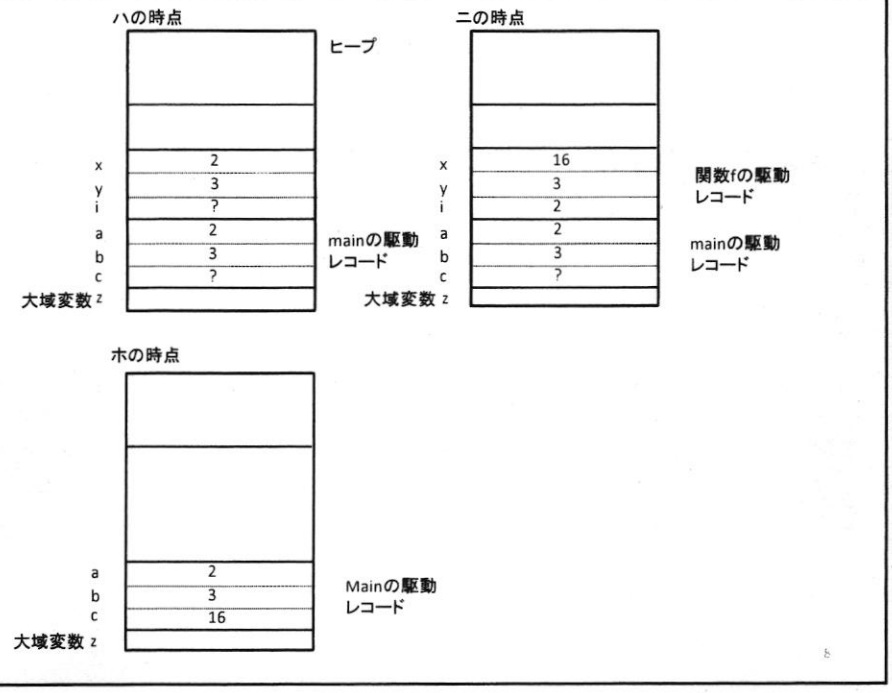


値呼びの実行時スタックの変化

関数 int f (int x, int y)
{ int i;
for (i=0;i++;i<y) { ハ
x=x*2; }
return x; ニ
}

呼び出し側
int z;
main () { int a,b,c;
a=2;
b=3; イ
c= f(a, b); ロ
ホ }





引数の結合(引数の渡し方)

仮引数と実引数の対応のつけ方

値呼び call by value

呼び出し側は実引数を(その環境で評価し)、この値で手続き内の仮引数を初期化する

仮引数は手続き内では，ローカル変数として使用される

手続きから呼び側へのデータの流れは無い

参照呼び call by reference

呼び出し側は実引数のアドレス(相対番地)を手続きに渡す

渡された番地は呼び側と手続きで共有する

名前呼び call by name

仮引数は手続きのローカル変数であるが，呼び出し側の環境内にあるロケーションを表す

仮引数の出現位置に対して対応する実引数を(評価しない)その字面のままで置き換え、その後に実行する

値呼び

```
関数  int f (int x, int y)          ハ          呼び出し側
{       int i;                                   a=2;
        for (i=0;i++;i<y) {                      b=3;          イ
          x=x*2; }                               c= f(a, b);   ロ
        return x;                   ニ                        ホ
}
```

イの時点　a [2] b [3] c [?]

ロの時点(関数が呼び出される)　x [?] y [?] i [?]　a [2] b [3] c [?]

仮引数x, yのメモリ領域が用意される

ハの時点　x [2] y [3] i [?]　a [2] b [3] c [?]

実引数の値が仮引数のロケーションに渡される

ニの時点　x [16] y [3] i [2]　a [2] b [3] c [?]

ホの時点　a [2] b [3] c [16]

関数f()が戻ると、メモリ領域が無くなる

参照呼び call by reference

Pascalプログラム

```
program callbyvar (input, output);
 var max, min:integer;
procedure swap(var x, y:integer);
 var temp:integer;
 begin temp:=x; x:=y; y:=temp end;

begin
 read (min, max);
 if min>max then swap (min, max);
end.
```

参照呼び

等価なCプログラム

```
void swap (int *x, int *y) {
int temp;
temp=*x;
*x=*y;
*y=temp; }
```

C言語にはないがポインタが近い

テキストには記載されていません

名前呼び call by name

例：ALGOLの手続き

```
procedure P(A, B, C, D); real A, B, C, D;
begin
    A:=A+1; B:=B+D; C:=C+D*D;
end;
```

ここで，P(x, y, z, u+2)を呼ぶと，仮引数を実引数に置き換えて，
x:=x+1; y:=y+u+2; z:= z+(u+2)*(u+2)　を実行する

名前呼びは強力な機能であるが，実現が容易でないので，
ALGOLで採用されたたが，その他の言語では使わなくなった

問題

演習【9】

例4.15中の分の並びi:=2; A[2]:=3; A[3]:=4; swap (i, A[i]);
の実行において，手続きを値呼び出し，参照呼出し，名前呼び出しで実行したとき，実行結果を比較せよ.

```
procedure swap (x, y) : integer x, y;
 begin integer temp;
   temp:=x; x:=y. y:=temp
 end;
```

13

名前の有効範囲(スコープ)

名前の有効範囲は、静的有効範囲である

プログラムの文面上の宣言文の位置と構造に依存

C言語の名前の有効範囲

C言語は分割コンパイルが可能で、複数個の<コンパイル単位>から構成される

<コンパイル単位> ::= <外部宣言>*

<外部宣言> := <関数定義> | <宣言>

別のコンパイル単位の大域変数と関数定義を利用するには、extern宣言

別のコンパイル単位で、変数と関数定義を利用させないようにするには、static宣言

14

C言語の変数名の有効範囲

- 変数名を関数定義の外で定義することで、大域変数となる
- 関数の仮引数、関数内の複文の始めに定義された変数名は局所的変数となる
- 変数名をextern宣言することで、他のコンパイル単位で定義された大域変数名を利用できる
- 大域変数名の有効範囲
  - コンパイル単位内の定義(の直後からコンパイル単位の終わりまで
  - ただし、その有効範囲内に同じ名前の変数が局所変数として定義されている場合は、内側の局所変数の有効範囲が優先される
- 局所変数名の有効範囲は、複文の先頭で定義(または宣言)されるときはその直後から複文の終わりまで
- 手続きの仮引数の有効範囲はその手続き本体

C言語の関数名の有効範囲

- 関数名は大域的名前である
  - C言語では，関数の中で関数を定義できない
- 関数名をextern宣言、またはプロトタイプ宣言することで、他のコンパイル単位で定義された関数を利用できる
  - 同じコンパイル単位内で定義される関数名をその定義に先行して，プロトタイプ宣言できる
- 関数名の有効範囲はそのコンパイル単位内の宣言の直後からコンパイル単位の終わりまで
  - ただし、その有効範囲内に同じ名前の変数が局所変数として定義されているならば、局所変数の有効範囲が優先される

変数の存続範囲

変数の存続範囲

変数に対してメモリ領域を割り当てている期間

C言語の変数名の存続範囲

- 大域的な変数の存続範囲は，プログラムの開始から終了までの期間
- 局所変数の存続範囲は，その局所変数が定義されている関数が呼び出されて，関数本体の実行開始から終了までの期間
  - なお，関数が再帰的に定義されているときは，自分自身を呼び出すことにより，同じ局所変数に新たなメモリ領域が割り当てられる
- static(静的)な変数として定義すると、プログラムの開始時にメモリ領域を一度割り当てられるだけで，自分自身の関数を呼び出されても新たにメモリ領域は割り当てられない
  - 静的変数は大域変数とほぼ同じ

問題

演習【10】

例4.17の表4.1のプログラムを実行した時の、出力を与えよ。

```
int x=5;
int  f (int y) {return x+y;}
int main () {
  int x, y;
  x=6;
  { int x; x=7; y=f(x); }
    printf ("%d\n, x+y);
}
```

再帰呼び出し

再帰呼び出し

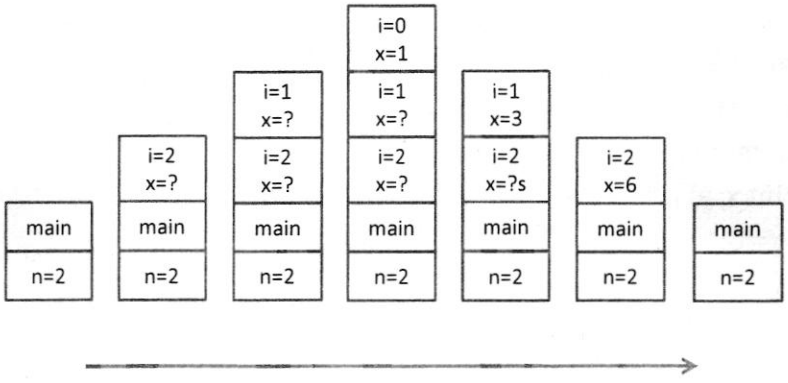
手続きの実行中に自分自身の手続きを呼び出す

例　再帰的関数

f(n) = if (n>1) then n* f(n-1) else 1

19

実行時スタックの変化



20

計算機言語講義資料 第4回 補足
命令型プログラミング言語(2)

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

その他の話題

手続きに関する話題(その1)

参照呼びの場合の手続き解消

手続きの呼び出しのところに，呼び出される手続きの本体を埋め込むことで，呼び出しを無くすこと

```
program A (input, output);
var a, b,c :intger;
    K:array of [1..10] of integer;
procedure arrange (var x, y:integer);
 var z : integer;
 begin
   if x<y then
   begin
     z:=x; x:=y; y:=z;
   end
 end
begin
 arrange(a, K[b]);
 arrange (b,c);
end .
```

参照呼び

⟺

```
program A (input, output);
var a, b,c :intger;
    K:array of [1..10] of integer;
begin
 if a<K[b] then begin d:=a; a:=K[b];
          K[b]:=d end;
 if b<c then begin d:=b; b:=c; c:=d end;
end.
```

テキストには記載されていません

手続きに関する話題(その2)

値呼びの場合の手続き解消

```
program A (....);
var a, b,c, d :integer;
 procedure P (x, y,z :integer; var
u:integer);
begin
   u:=x+y*z;
  end
begin
 P(a+b, a-b, c, d);
 P(a+b*b; c, 11, a);
end .
```

値呼び

⟺ 等価

```
program A (input, output);
var a, b,c, d, x, y, z :integer;
begin
 x:=a+b; y:=a-b; z:=c; d:=x+y*z;
 x:=a+b*b; y:=c; z:=11; a:=x+y*z;
end.
```

テキストには記載されていません

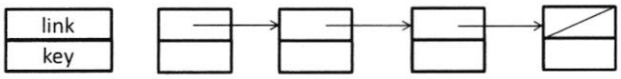
4

再帰呼び出しの解消 (その1)

再帰呼び出しの無い形に等価変換する

再帰呼び出しの例

```
struct node {int key;  struct node *link; };
struct node  *look (int i; struct node *ptr) {
 if (ptr==NULL) {  return (NULL); }
 else if (ptr->key == i) {
   return (ptr); }
 else {
   return (look (i, ptr->link)); }
}
```



上記の線形リスト中にkey iを持つレコードを探し，あればそのレコードを指すポインタを返し，無い場合はNULLを返す
最後に自分を呼ぶという形tail recursionになっている

テキストには記載されていません

再帰呼び出しの解消 (その2)

tail recursion(末尾再帰形)になっている再帰呼び出しは，goto文を用いることで再帰呼び出しを無くすことができる

goto文を用いて，再帰呼び出しを解消した例

```
struct node {int key;  struct node *link; };
struct node  *look (int i; struct node *ptr) {
 l: if (ptr==NULL) {  return (NULL); }
 else if (ptr->key == i) {
   return (ptr); }
 else {
   ptr=ptr->link;
   goto l; }
}
```

テキストには記載されていません

再帰呼び出しの解消 (その3)

goto文を用いないで，再帰呼び出しを解消した例

```
struct node {int key;  struct node*link; };
struct node  *look (int i; struct node *ptr) {
 int found = 0; /* 0:FALSE */
 while ((found==0)&(ptr!=NULL)  {
  if (ptr->key == i) {
     found=1; }
  else {  ptr = ptr->link; }
 }
 return (ptr);
}
```

テキストには記載されていません

再帰呼び出しの解消 (その4)

一般的には

末尾再帰形

```
A(x) = if p(x) then a(x)
       else if q(x) then b(x)
            else A(f(x))
```

goto文を用いた解消

```
A(x) : l: = if p(x) then A:=a(x)
       else if q(x) then A:=b(x)
            else x:=f(x); goto l
             end;
```

goto文を用いない解消

```
A(x) : notyet := true
      while notyet and not  p(x) do
        if q(x) then notyet := false else x:= f(x);
      A := if p(x) then a(x) else b(x)
```

テキストには記載されていません

再帰呼び出しの解消 (その5)

可換，かつ，結合則の成立する場合

```
A(x) = if p(x) then a(定数)
       else x・A(f(x))
```

ただし，・は可換，結合的である

・・・ (1)

次のように変換できる

```
A(x) : var y
       y := a;
       while not p(x) do
       begin
        y := x・y;
        x := f(x);
       end;
       A := y;
```

・・・ (2)

テキストには記載されていません