コンパイラ

第13回 実行時環境

— 変数と関数 —

http://www.info.kindai.ac.jp/compiler

E館3階E-331 内線5459

takasi-i@info.kindai.ac.jp

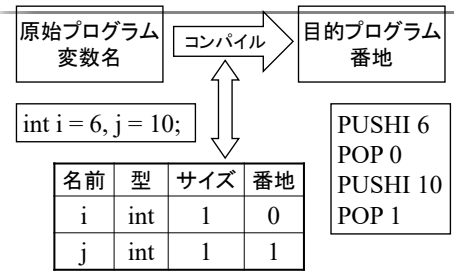
1

コンパイラの構造

- 字句解析系
- 構文解析系
- 制約検査系
- 中間コード生成系
- 最適化系
- 目的コード生成系

2

変数の番地



名前	型	サイズ	番地
i	int	1	0
j	int	1	1

変数には(仮想)メモリ上の番地を割り当てる

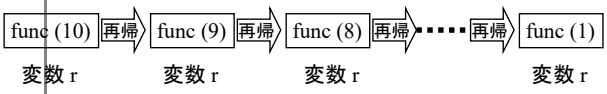
3

番地の割り当て

```
int func (int i) {
    int r = 1;
    if (i > 1) r += func (i-1);
    return r;
}
```

再帰

変数 r の番地は？



変数 r　変数 r　変数 r　変数 r

- 変数 r には複数の番地が必要
- コンパイル時には必要な番地の個数は不明

番地の静的な割り当ては不可能

4

変数の番地

- 静的番地 (static address)
  - コンパイル時に番地を決定可能
    - 大域変数, 静的局所変数等
- 動的番地(dynamic address)
  - 実行時に呼び出すまで分からない
  - 呼び出されたときに番地を決定
  - 静的リンク(static link)と相対番地を用いて管理

(※) 変数の番地は動的だがスコープは静的

    - 局所変数

5

変数の種別

	割当	有効範囲 (スコープ)	
大域変数 (global)	静的	プログラム全体	ブロック外で宣言 全てのブロックで共通して使用
静的局所変数 (static local)	静的	ブロック内	ブロック内で宣言 1度だけ作られる ブロック間で共通して使用
局所変数 (local)	動的	ブロック内	ブロック内で宣言 ブロックが呼ばれる度に作られる ブロック毎に領域を確保
局所変数	動的	任意	解放位置をプログラム中に記述

6

スコープルール(scope rule)

- スコープルール
  - 名前の有効範囲

```
if (a == 0) {
    int x;
     :
}
for (int i=0; i<10; ++i) {
     :
}
```

int 型変数 x は
この内部のみで有効

int 型変数 i は
この内部のみで有効

有効範囲ごとに記号表を作成する

7

スコープルール

- 記号表の動的管理
  - ブロックに入る → 新しい記号表を作成
  - ブロックから出る → 最新の記号表を削除
- 名前の参照
  - 最も新しい記号表から順に検索

8

記号表の動的管理

```
{
  int i, j;
  {
    int k, l;
  }
   :
  {
    int m, n;
  }
   :
}
```

i	変数	int	0
j	変数	int	1

9

記号表の動的管理

```
{
  int i, j;
  {
    int k, l;
  }
   :
  {
    int m, n;
  }
   :
}
```

i	変数	int	0
j	変数	int	1

k	変数	int	2
l	変数	int	3

10

記号表の動的管理

```
{
  int i, j;
  {
    int k, l;
  }
   :
  {
    int m, n;
  }
   :
}
```

i	変数	int	0
j	変数	int	1

~~k 変数 int 2~~
~~l 変数 int 3~~

11

記号表の動的管理

```
{
  int i, j;
  {
    int k, l;
  }
   :
  {
    int m, n;
  }
   :
}
```

i	変数	int	0
j	変数	int	1

~~k 変数 int 2~~
~~l 変数 int 3~~

m	変数	int	2
n	変数	int	3

k と m, l と n で
共通の領域を使用

12

記号表の動的管理

```
{
   int i, j;
   {
      int k, l;
   }
    :
   {
      int m, n;
   }
    :
}
```

i	変数	int	0
j	変数	int	1

~~k~~	~~変数~~	~~int~~	~~2~~
~~l~~	~~変数~~	~~int~~	~~3~~

~~m~~	~~変数~~	~~int~~	~~2~~
~~n~~	~~変数~~	~~int~~	~~3~~

13

名前の参照

```
{
   int i, j, k;
   {
      int i, j;
      {
         int i
      }
      :
   }
   :
}
```

i	変数	int	0
j	変数	int	1
(k)	変数	int	2

i	変数	int	3
(j)	変数	int	4

(i)	変数	int	5

変数 i の参照
変数 j の参照
変数 k の参照

最新の表から順に参照

14

名前の参照

```
{
   int i, j, k;
   {
      int i, j;
      {
         int i
      }
      :
   }
   :
}
```

レベル	変数		
2	i	j	k
1	i	j	
0	i		

レベルの若いものから参照

15

関数呼び出し

```
main () {
   int a, b, c;
      :
   c = func (a);
      :
}

func (int x) {
   int i;
     :
   return i;
}
```

~~a~~	~~変数~~	~~int~~	~~3~~
b	変数	int	4
c	変数	int	5

a	変数	int	3
b	変数	int	4
c	変数	int	5

x	引数	int	6
i	変数	int	10

16

動的変数の番地の割り当て

- ~~new (Java)~~
  - new : 番地割り当て命令

```
{
   int a[];
      :
   a = new int [100];
      :
}
```

この時点では番地未定

メモリ確保 int [100]

メモリ解放

メモリ確保位置をプログラマが指定
メモリ解放位置はブロック終了時

17

記号表の動的管理

```
{
   int a[], b[];
      :
   b = new int [100];
      :
   a = new int [200];
      :
}
```

b の有効範囲
a の有効範囲

a	変数	int[]	未定
b	変数	int[]	未定

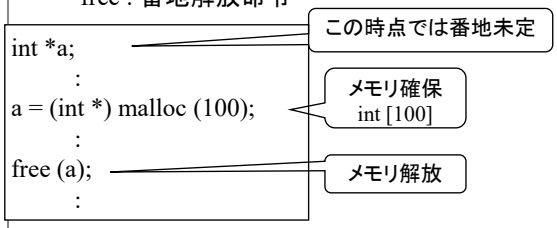
a	変数	int[]	未定
b	変数	int[]	0〜99

a	変数	int[]	100〜299
b	変数	int[]	0〜99

18

動的変数の番地の割り当て

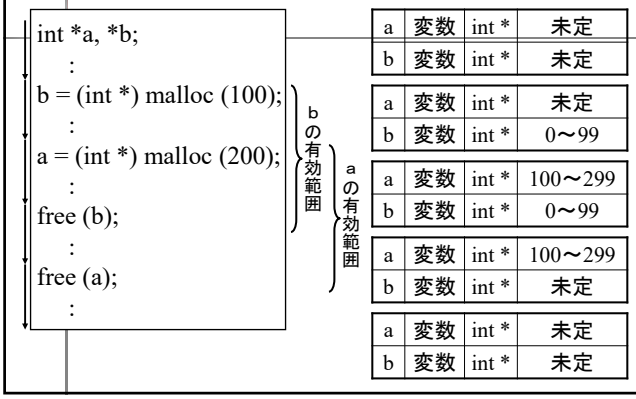
- ~~malloc と free (C言語)~~
  - malloc : 番地割り当て命令
  - free : 番地解放命令



メモリ確保位置、解放位置をプログラマが指定

19

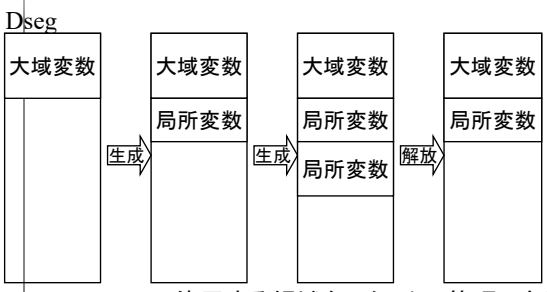
記号表の動的管理



20

動的変数の管理

- ブロック終了時にメモリ解放される場合

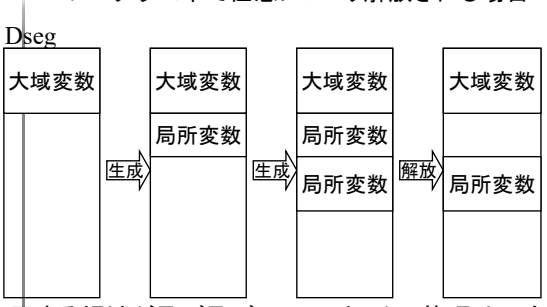


使用する領域をスタックで管理できる

21

動的変数の管理

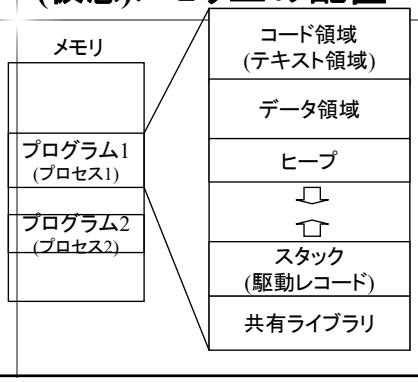
- プログラム中で任意にメモリ解放される場合



使用する領域が飛び飛びに ⇒ スタックで管理は不向き

22

プログラム(プロセス)の(仮想)メモリ上の配置



23

プログラム(プロセス)の構造

- コード領域(code segment),
  テキスト領域(text segment)
  - プログラム命令のコード
    (歴史的な理由からテキストと呼ばれている)
- データ領域(data segment)
  - 静的なデータ

24

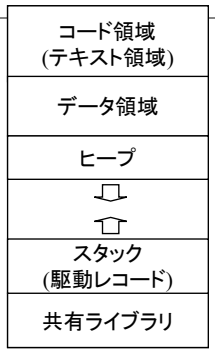
プログラム(プロセス)の構造

- ヒープ(heap)
  - プログラム実行時に確保されるメモリ領域
- スタック(stack)
  駆動レコード(activation record)
  スタックフレーム(stack frame)
  - 関数の引数, 関数の局所変数, 前フレームへのポインタ, 関数呼び出しの戻り番地

25

変数の記憶領域

変数	記憶領域
大域変数	データ領域 or スタックの底部
局所変数 (ブロック終了時に解放)	スタック
局所変数 (任意に解放)	ヒープ



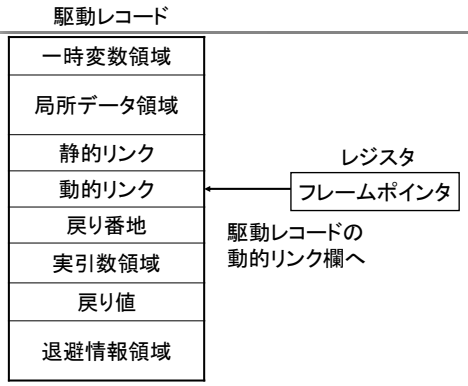
26

駆動レコード(activation record) スタックフレーム(stack frame)

- 駆動レコード, スタックフレーム
  - 関数・手続きの実行に必要な情報を格納
    - 関数・手続きの引数, 戻り番地, 局所変数へのポインタ等
  - 関数・手続き呼び出し時
    - 新たな駆動レコード作成、スタックに積む
  - 関数・手続き完了時
    - 駆動レコードをスタックから取り去る
- フレームポインタ (frame pointer)
  - 現在実行中の駆動レコードへのポインタ
  - レジスタに格納されることが多い

27

駆動レコードとフレームポインタ



28

駆動レコード

- 一時変数領域 (temporal variable area)
  - 作業用領域
- 局所データ領域 (local data area)
  - 局所変数等

29

駆動レコード

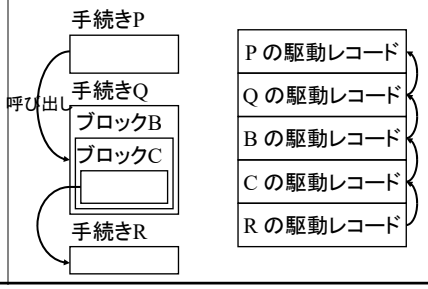
- 静的リンク(static link)
  アクセスリンク(access link)
  - 親ブロックを指すポインタ
- 動的リンク(dynamic link)
  制御リンク(control link)
  - 手続きの呼び出し元を指すポインタ

30

駆動レコード

- 静的リンク(static link)

  アクセスリンク(access link)
  - 親ブロックを指すポインタ

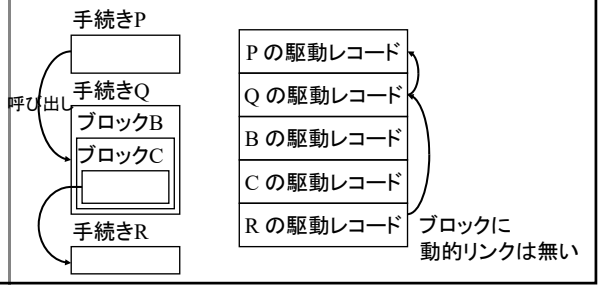


31

駆動レコード

- 動的リンク(dynamic link)

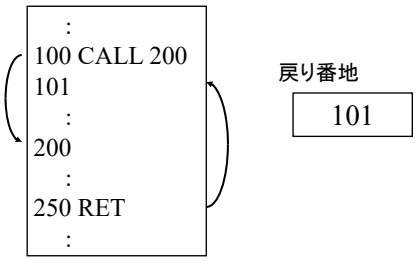
  制御リンク(control link)
  - 手続きの呼び出し元を指すポインタ



32

駆動レコード

- 戻り番地(return address)
  - 呼び出しから戻るときの番地



33

駆動レコード

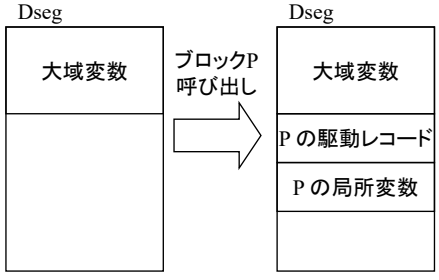
- 実引数(actual parameters)
  - 呼び出し元の引数の値を格納
- 戻り値(return value)
  - 呼び出し元へ返す値を格納
- 退避情報領域
  - 呼び出し元のレジスタの値等

```
main () {        実引数
   :
   x = func (1, 2);
   :
}
                 仮引数
func (int i, j) {
   :
   return a;
}
                 戻り値
```

34

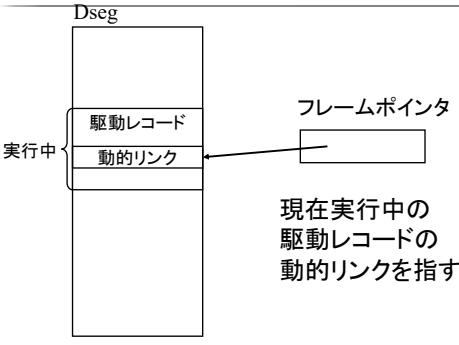
駆動レコードの記憶

- 駆動レコードは Dseg に記憶



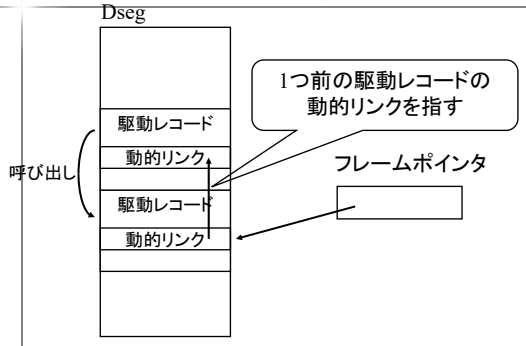
35

フレームポインタ (frame pointer)



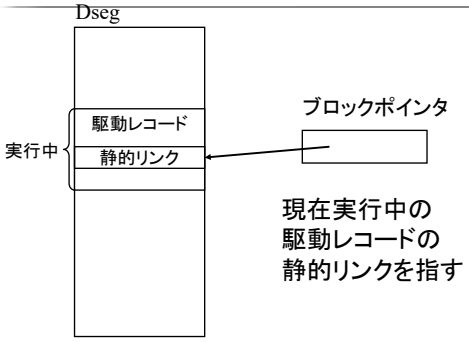
36

フレームポインタ (frame pointer)



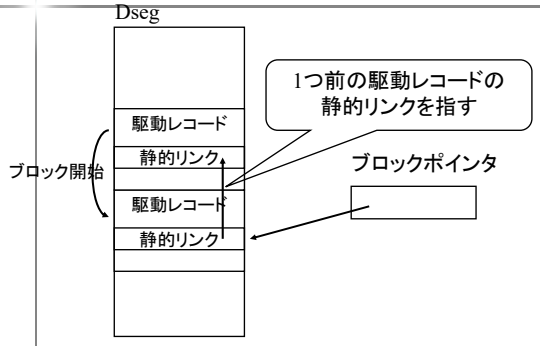
37

ブロックポインタ (block pointer)



38

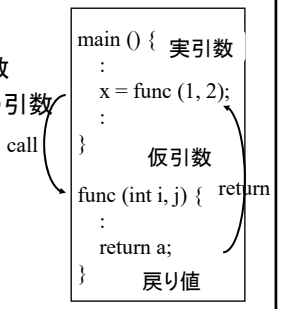
ブロックポインタ (block pointer)



39

関数呼び出し

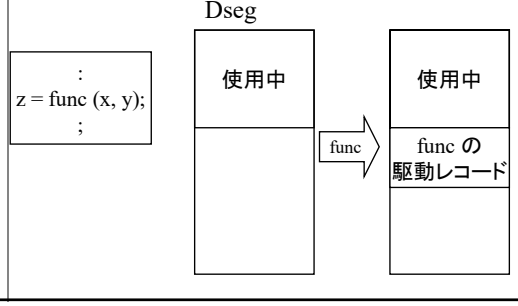
- 関数呼び出し (call)
- 呼び出し元に戻る (return)
- 仮引数 : 関数定義時の引数
- 実引数 : 関数呼び出し時の引数



40

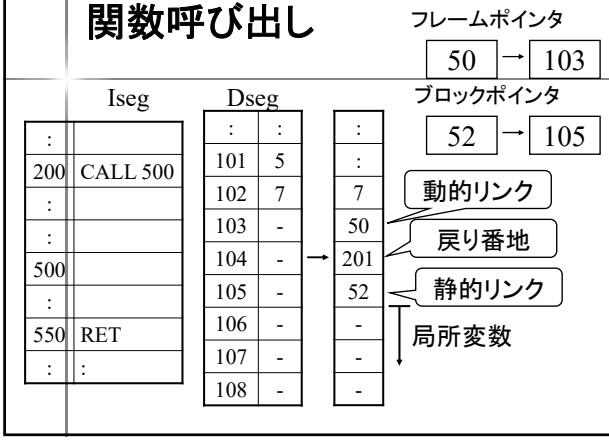
関数呼び出し

- 駆動レコードを生成, Dseg に加える



41

関数呼び出し



42

オペランド無しのPOP命令

- POP *d*
  - スタックの値を Dseg の *d* 番地に格納
- POP
  - スタックの値を Dseg の最後の番地に格納

データポインタ
Dseg の最後の番地を指すポインタ

43

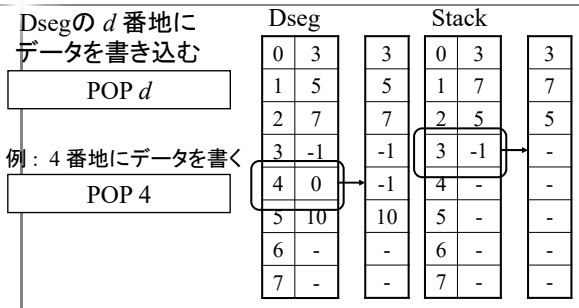
Stack → Dseg POP 命令

Dsegの *d* 番地にデータを書き込む

POP *d*

例：4 番地にデータを書く

POP 4



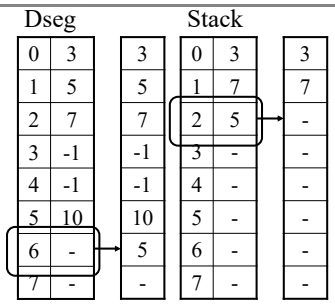
44

Stack → Dseg POP 命令

Dsegの最後の番地にデータを書き込む

POP

データポインタ 5 → 6



45

CALL, RET

- CALL 命令
  - CALL *addr* (飛び先番地)
- RET 命令
  - RET *num* (引数の個数)

CALL

```
Dseg [++DP] := FP;
FP := DP;
Dseg [++DP] := PC + 1;
Dseg [++DP] := BP;
BP := DP;
PC := addr;
```

RET

```
BP := Dseg [FP+2];
PC := Dseg [FP+1];
DP := FP - num - 1;
FP := Dseg [FP];
```

46

CALL命令

200 CALL 500　プログラムカウンタ 200 → 500

```
Dseg [++DP] := FP;
FP := DP;
Dseg [++DP] := PC + 1;
Dseg [++DP] := BP;
BP := DP;
PC := addr;
```

フレームポインタ 50 → 103
ブロックポインタ 52 → 105
データポインタ 102 → 105

47

RET命令

550 RET 1　プログラムカウンタ 550 → 201

```
BP := Dseg [FP+2];
PC := Dseg [FP+1];
DP := FP - num - 1;
FP := Dseg [FP];
```

フレームポインタ 103 → 50
ブロックポインタ 105 → 52
データポインタ 107 → 101

48

関数呼び出し

引数無しの関数呼び出し　引数ありの関数呼び出し

```
func ();
```

```
CALL 200
    :
RET 0;
```

CALL 直前に
引数の値を Dseg に積む

```
func (5, 10);
```

```
PUSHI 5
POP
PUSHI 10
POP
CALL 300
    :
RET 2;
```

オペランド無しの POP

引数の個数

49

戻り値

戻り値無し　戻り値あり

```
return;
```

```
CALL 200
    :
RET 1;
```

RET 直前に
戻り値をスタックに積む

```
return 1;
```

```
CALL 200
    :
PUSHI 1
RET 1;
```

戻り値

引数の個数

50

START, HALT

- START 命令
  - START
- HALT 命令
  - HALT

```
main () {
    :
}
```

START

HALT

START

```
Dseg [++DP] := -1;
FP := DP;
Dseg [++DP] := -1;
Dseg [++DP] := -1;
BP := DP;
```

main 関数なので
呼び出し元無し

HALT

システムに制御を返す

51

START命令

```
0 START
```

プログラムカウンタ 0 → 1

```
Dseg [++DP] := -1;
FP := DP;
Dseg [++DP] := -1;
Dseg [++DP] := -1;
BP := DP;
```

Dseg

0	-
1	-
2	-
3	-
4	-
5	-
6	-

-1	動的リンク
-1	戻り番地
-1	静的リンク
-	
-	
-	
-	

フレームポインタ - → 0
ブロックポインタ - → 2
データポインタ -1 → 2

52

BEGIN, END

- BEGIN 命令
  - BEGIN
- END 命令
  - END

```
{
    :
}
```

BEGIN

END

BEGIN

```
Dseg [++DP] := BP;
BP := DP;
```

END

```
DP := BP-1;
BP := Dseg [BP];
```

53

BEGIN命令

```
300 BEGIN
```

プログラムカウンタ 300 → 301

```
Dseg [++DP] := BP;
BP := DP;
```

Dseg

:	:
101	5
102	7
103	-
104	-
105	-
106	-
107	-
108	-

:
5
7
52
-
-
-
-
-

静的リンク

フレームポインタ 50 → 50
ブロックポインタ 52 → 103
データポインタ 102 → 103

54

相対番地(relative address)

- 相対番地
  - ブロックの先頭を基準とした番地
    - (ブロック番号, 先頭からの相対位置)
  - 実番地は実行時に計算

```
{
  int i, j;          // ブロック2
  {
    int x, y;        // ブロック1
    {
      int a, b, c;   // ブロック0
    }
  }
}
```

変数	相対番地	実番地
i	(2, 1)	3
j	(2, 2)	4
x	(1, 1)	7
y	(1, 2)	8
a	(0, 1)	10
b	(0, 2)	11
c	(0, 3)	12

55

名前の参照

```
{
  int a,x;
    :
  {
    int u,a;
    :
  }
  :
}
```

ブロック内番地は1番地から

変数名	レベル	ブロック内番地	相対番地
a	0	1	0,1
x	0	2	0,2

変数名	レベル	ブロック内番地	相対番地
a	0	1	-
x	0	2	1,2
u	1	1	0,1
a	1	2	0,2

56

相対番地

- 相対番地
  - ブロックの先頭を基準とした番地
  - 実番地は実行時に計算する

```
int i, j;          // ブロック2
{
  int x, y;        // ブロック1
  {
    int a, b, c;   // ブロック0
  }
}
```

Dseg

変数	実番地	相対番地	
DL	0	(2, -2)	
RA	1	(2, -1)	
SL	2	(2, 0)	ブロック2先頭
i	3	(2, 1)	ブロック2
j	4	(2, 2)	
SL	6	(1, 0)	ブロック1先頭
x	7	(1, 1)	ブロック1
y	8	(1, 2)	
SL	9	(0, 0)	ブロック0先頭
a	10	(0, 1)	ブロック0
b	11	(0, 2)	
c	12	(0, 3)	

57

相対番地

- ブロック番号とブロック内番地で記述

例 : (0, 5)　(ブロック番号, ブロック内番地)

ブロック番号	変数の種別
-1	大域変数
0	現在のブロックの局所変数
1	1つ上のブロックの局所変数
2	2つ上のブロックの局所変数
:	:

58

相対番地

- ブロック番号とブロック内番地で記述

例 : (0, 5)　(ブロック番号, ブロック内番地)

ブロック内番地	変数の種別
〜-3	引数
-2	動的リンク
-1	戻り番地
0	静的リンク
1〜	局所変数

59

番地の計算

- 動的番地
  - アセンブリコードでは相対番地で記述
  - 実行時に実番地を計算

```
相対番地                    実番地
PUSHI (0, 1)      実行時    PUSHI 50
PUSH  (1, 3)      ----->    PUSH  22
ASSIGN                      ASSIGN
```

60

番地の計算

- ea (-1, a) = -1 + a　　// 大域変数
- ea (0, a) = BP + a　　// ブロック内変数
- ea (1, a) = Dseg [BP] + a　// 親ブロック内変数
- ea (2, a) = Dseg [Dseg [BP]] + a
- ea (3, a) = Dseg [Dseg [Dseg [BP]]] + a
- ea (m, a) = (静的リンクを m 回辿る) + a

61

番地の計算 (ブロック内変数の場合)

```
{
   int x, y, z;
   :
}
```

x : (0, 1)

ea (0, 1) = BP + 1
= 101

ブロックポインタ 100

Dseg

変数	相対番地	実番地	値
SL	(0, 0)	100	50
x	(0, 1)	101	2
y	(0, 2)	102	10
z	(0, 3)	103	30
:	:	:	:

62

番地の計算 (1つ上のブロック内変数の場合)

```
{
   int i, j;
   {
      :
   }
   :
}
```

j : (1, 2)

ea (1, 2)
= Dseg[BP] + 2
= 52

ブロックポインタ 100

Dseg

変数	相対番地	実番地	値
SL	(1, 0)	50	20
i	(1, 1)	51	5
j	(1, 2)	52	10
:	:	:	:
SL	(0, 0)	100	50
:	:	:	:

63

番地の計算 (大域変数の場合)

```
int n, m;
main() {
   :
}
```

n : (-1, 1)

ea (-1, 1) = -1 + 1
= 0

ブロックポインタ 100

Dseg

変数	相対番地	実番地	値
n	(-1, 1)	0	4
m	(-1, 2)	1	6
:	:	:	:
SL	(0, 0)	100	60
:		:	:

64

番地の計算

```
int n=2, m=3;
main () {
   int i=5, j=10;
      :
   {
      int x=4, y=8;
   }
      :
}
```

Dseg

	変数	相対番地	実番地	値
大域	n	(-1, 1)	0	2
大域	m	(-1, 2)	1	3
外部ブロック	DL	(1, -2)	2	-1
外部ブロック	RA	(1, -1)	3	-1
外部ブロック	SL	(1, 0)	4	-1
外部ブロック	i	(1, 1)	5	5
外部ブロック	j	(1, 2)	6	10
内部ブロック	SL	(0, 0)	7	4
内部ブロック	x	(0, 1)	8	4
内部ブロック	y	(0, 2)	9	8

65

番地の計算 (引数の場合)

```
func (int s, int t) {
   int i,j;
      :
}
```

s : (0, -4)

ea (0, -4)
= BP - 4
= 96

ブロックポインタ 100

Dseg

変数	相対番地	実番地	値
:	:	:	:
s	(0, -4)	96	3
t	(0, -3)	97	6
DL	(0, -2)	98	48
RA	(0, -1)	99	201
SL	(0, 0)	100	50
i	(0, 1)	101	3
j	(0, 2)	102	4
:	:	:	:

66

番地の計算

```
func (int x, int y) {
  int s=4, t=8;
    :
}

main () {
  int i=5, j=10;
    :
  func (20, 30);
    :
}
```

	変数	相対番地	実番地	値
main	DL	(1, -2)	0	-1
	RA	(1, -1)	1	-1
	SL	(1, 0)	2	-1
	i	(1, 1)	3	5
	j	(1, 2)	4	10
func	x	(0, -4)	5	20
	y	(0, -3)	6	30
	DL	(0, -2)	7	0
	RA	(0, -1)	8	100
	SL	(0, 0)	9	2
	s	(0, 1)	10	4
	t	(0, 2)	11	8

67

コード生成の例

```
int n, m;
main () {
  int i, j;
    :
  {
    int x, y, z;
    z = 1;     // ブロック内変数
    j = 10;    // 親ブロック内変数
    n = 100; // 大域変数
  }
    :
}
```

```
z = 1:
  PUSHI (0, 3)
  PUSHI 1
  ASSGN
  REMOVE
j = 10:
  PUSHI (1, 2)
  PUSHI 10
  ASSGN
  REMOVE
n = 100:
  PUSHI (-1, 1)
  PUSHI 100
  ASSGN
  REMOVE
```

68

コード生成の例

```
main () {
  int i, j;
    :
  i = func (10, 20);
    :
}

int func (int x, int y) {
  int z;
  z = x;
  return z;
}
```

```
i = func (10, 20):
  0 PUSHI (0, 1)
  1 PUSHI 10
  2 POP
  3 PUSHI 20
  4 POP
  5 CALL 50
  6 ASSGN
  7 REMOVE
    :
z = x:
  50 PUSHI (0, 1)
  51 PUSHI (0, -4)
  52 ASSIGN
  53 REMOVE
return z:
  54 PUSHI (0, 1)
  55 RET 2
```

69

引数の引渡し

- 値渡し (call by value, pass by value)
- 結果渡し (call by result, pass by result)
- 値結果渡し (call / pass by value-result)
- 参照渡し (call / pass by reference)
- 名前渡し (call / pass by name)

70

値渡し
(call by value, pass by value)

- 値渡し
  - 呼び出し時に実引数の値を仮引数にコピー

```
{
  int a=1, b=2;
  func (a, b);
  :
}

func (int i, int j) {
  :
```

i := a
j := b

```
PUSH (0, 1)
POP
PUSH (0, 2)
POP
CALL 100
```

71

値渡し

```
main () {
  int i=10, j=20;
  func (i, j);
}
func (int x, int y) {
  x = y;
}
```

変数	相対番地	実番地	値	
DL	(1,-2)	0	-1	
RA	(1,-1)	1	-1	
SL	(1, 0)	2	-1	
i	(1, 1)	3	10	
j	(1, 2)	4	20	値をコピー
x	(0,-4)	5	10	
y	(0,-3)	6	20	
DL	(0,-2)	7	0	
RA	(0,-1)	8	6	
SL	(0, 0)	9	2	

72

参照渡し
(call / pass by reference)

- 参照渡し
  - 呼び出し時に実引数の番地を仮引数にコピー

```
{
   int a=1, b=2;
   func (a, b);
   :
}
func (int i, int j) {
   :
```

i := &a
j := &b

```
PUSHI (0, 1)
POP
PUSHI (0, 2)
POP
CALL 100
```

73

参照渡し

```
main () {
   int i=10, j=20;
   func (i, j);
}
func (int x, int y) {
   x = y;
}
```

変数	相対番地	実番地	値
DL	(1,-2)	0	-1
RA	(1,-1)	1	-1
SL	(1, 0)	2	-1
i	(1, 1)	3	10
j	(1, 2)	4	20
x	(0,-4)	5	3
y	(0,-3)	6	4
DL	(0,-2)	7	0
RA	(0,-1)	8	6
SL	(0, 0)	9	2

番地をコピー

74

引数の参照

~~引数 x : (0, -3)~~

値渡しの場合

左辺値

```
PUSHI (0, -3)
```

右辺値

```
PUSH (0, -3)
```

変数と同様に処理

参照渡しの場合

左辺値

```
PUSH (0, -3)
```

右辺値

```
PUSH (0, -3)
LOAD
```

番地を値に変換する

75

プログラム例

```
main () {
   int i=10, j=20;
   func (i, j);
}
func (int x, int y) {
   x = y;
}
```

値渡しの場合

```
 0 START
 1 PUSHI 10
 2 POP (0, 1)
 3 PUSHI 20
 4 POP (0, 2)
 5 PUSH (0, 1)
 6 POP
 7 PUSH (0, 2)
 8 POP
 9 CALL 11
10 HALT
11 PUSHI (0, -4)
12 PUSH (0, -3)
13 ASSGN
14 REMOVE
15 RET 2
```

参照渡しの場合

```
 0 START
 1 PUSHI 10
 2 POP (0, 1)
 3 PUSHI 20
 4 POP (0, 2)
 5 PUSHI (0, 1)
 6 POP
 7 PUSHI (0, 2)
 8 POP
 9 CALL 11
10 HALT
11 PUSH (0, -4)
12 PUSH (0, -3)
13 LOAD
14 ASSGN
15 REMOVE
16 RET 2
```

76

値渡しと参照渡し

```
{
   int a = 1, b = 2;
   int c = func (a, b);
      :
}
func (int x, int y) {
   y = 2*y;
   x = x+y;
   return x;
}
```

func 呼び出し後の値

	a	b	c
値渡し	1	2	5
参照渡し	5	4	5

参照渡しは仮引数の値を変えると実引数の値も変わる

77

配列型引数

```
{
   int a[100];
      :
   func (a);
      :
}
func (int[] x) {
      :
}
```

値渡しの場合

```
PUSH (0, 1)
POP
PUSH (0, 2)
POP
   :
PUSH (0, 100)
POP
CALL 500
   :
RET 100
```

全ての要素をコピー

参照渡しの場合

```
PUSHI (0, 1)
POP
CALL 300
   :
RET 1
```

先頭の番地をコピー

多くの言語で配列は参照渡し

78

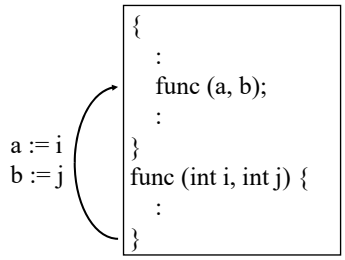
値渡しと参照渡しの長所と短所

- 値渡し
  - 関数間の独立性が高い
- 参照渡しの長所
  - 配列、オブジェクト等のデータ数の多いものでも速やかに渡せる

79

結果渡し
(call by result, pass by result)

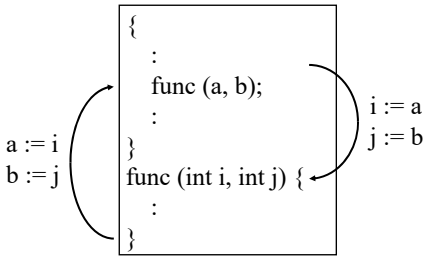
- 結果渡し
  - 完了時に仮引数の値を実引数にコピー



80

値結果渡し
(call / pass by value-result)

- 値結果渡し
  - 呼び出し時に実引数の値を仮引数にコピー
  - 完了時に仮引数の値を実引数にコピー



81

名前渡し
(call by name, pass by name)

- 名前渡し
  - 関数をマクロとして展開

```
{
  int a = 1, b = 2;
  func (a, b);
      :
}
func (int x, int y) {
  y = 2*y;
  x = x+y;
}
```

⇨

```
{
  int a = 1, b = 2;
  {
    b = 2*b;
    a = a+b;
  }
      :
}
```

x=a, y=b として func を展開

82

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j;
  return s;
}
int func2 (int x, int y) {
  int z;
  z = 5 * func1 (x, y);
  return z;
}
main () {
   int a = 20;
   print (func2 (n, a));
}
```

```
 0 PUSHI 50
 1 POP (-1, 1)
 2 JUMP 23
 3 PUSHI (0, 1)
 4 PUSH (0, -4)
 5 PUSH (0, -5)
 6 ADD
 7 ASSGN
 8 REMOVE
 9 PUSH (0, 1)
10 RET 2
11 PUSHI (0, 1)
12 PUSHI 5
13 PUSH (0, -4)
14 POP
15 PUSH (0, -3)
16 POP
```

```
17 CALL 3
18 MUL
19 ASSGN
20 REMOVE
21 PUSHI (0, 1)
22 RET 2
23 START
24 PUSHI 20
25 POP (0, 1)
26 PUSH (-1, 1)
27 POP
28 PUSH (0, 1)
29 POP
30 CALL 11
31 OUTPUT
32 HALT
```

83

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j; ③
  return s;
}
int func2 (int x, int y) {
  int z; ②
  z = 5 * func1 (x, y); ④
  return z;
}
main () {
   int a = 20; ①
   print (func2 (n, a)); ⑤
}
```

```
 0 PUSHI 50
 1 POP (-1, 1)
 2 JUMP 23
 3 PUSHI (0, 1)
 4 PUSH (0, -4)
 5 PUSH (0, -5)
 6 ADD
 7 ASSGN
 8 REMOVE ③
 9 PUSH (0, 1)
10 RET 2
11 ② PUSHI (0, 1)
12 PUSHI 5
13 PUSH (0, -4)
14 POP
15 PUSH (0, -3)
16 POP
```

```
17 CALL 3
18 MUL
19 ASSGN
20 REMOVE ④
21 PUSHI (0, 1)
22 RET 2
23 START
24 PUSHI 20 ①
25 POP (0, 1)
26 PUSH (-1, 1)
27 POP
28 PUSH (0, 1)
29 POP
30 CALL 11
31 OUTPUT ⑤
32 HALT
```

84

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j;
  return s;
}
int func2 (int x, int y) {
  int z;
  z = 5 * func1 (x, y);
  return z;
}
main () {
   int a = 20; ①
   print (func2 (n, a));
}
```

地点①実行時

変数	相対番地	実番地	値	変数	相対番地	実番地	値
~~n~~	~~-1,1~~	~~0~~	~~50~~			~~10~~	~~-~~
DL	0,-2	1	-1			11	-
RA	0,-1	2	-1			12	-
SL	0,0	3	-1			13	-
a	0,1	4	20			14	-
		5	-			15	-
		6	-			16	-
		7	-			17	-
		8	-			18	-
		9	-			19	-

FP 1 BP 3

85

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j;
  return s;
}
int func2 (int x, int y) {
  int z; ②
  z = 5 * func1 (x, y);
  return z;
}
main () {
   int a = 20; ①
   print (func2 (n, a));
}
```

地点②実行時

変数	相対番地	実番地	値	変数	相対番地	実番地	値
~~n~~	~~-1,1~~	~~0~~	~~50~~	~~z~~	~~0,1~~	~~10~~	~~-~~
DL	1,-2	1	-1			11	-
RA	1,-1	2	-1			12	-
SL	1,0	3	-1			13	-
a	1,1	4	20			14	-
x	0,-4	5	50			15	-
y	0,-3	6	20			16	-
DL	0,-2	7	1			17	-
RA	0,-1	8	31			18	-
SL	0,0	9	3			19	-

FP 7 BP 9

86

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j; ③
  return s;
}
int func2 (int x, int y) {
  int z; ②
  z = 5 * func1 (x, y);
  return z;
}
main () {
   int a = 20; ①
   print (func2 (n, a));
}
```

地点③実行時

変数	相対番地	実番地	値	変数	相対番地	実番地	値
~~n~~	~~-1,1~~	~~0~~	~~50~~	~~z~~	~~1,1~~	~~10~~	~~-~~
DL	2,-2	1	-1	i	0,-4	11	50
RA	2,-1	2	-1	j	0,-3	12	20
SL	2,0	3	-1	DL	0,-2	13	7
a	2,1	4	20	RA	0,-1	14	18
x	1,-4	5	50	SL	0,0	15	9
y	1,-3	6	20	s	0,1	16	70
DL	1,-2	7	1			17	-
RA	1,-1	8	31			18	-
SL	1,0	9	3			19	-

FP 13 BP 15

87

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j; ③
  return s;
}
int func2 (int x, int y) {
  int z; ②
  z = 5 * func1 (x, y); ④
  return z;
}
main () {
   int a = 20; ①
   print (func2 (n, a));
}
```

地点④実行時

変数	相対番地	実番地	値	変数	相対番地	実番地	値
~~n~~	~~-1,1~~	~~0~~	~~50~~	~~z~~	~~0,1~~	~~10~~	~~350~~
DL	1,-2	1	-1			11	50
RA	1,-1	2	-1			12	20
SL	1,0	3	-1			13	7
a	1,1	4	20			14	18
x	0,-4	5	50			15	9
y	0,-3	6	20			16	70
DL	0,-2	7	1			17	-
RA	0,-1	8	31			18	-
SL	0,0	9	3			19	-

FP 7 BP 9

88

プログラム例

```
int n = 50;
int func1 (int i, int j) {
  int s;
  s = i+j; ③
  return s;
}
int func2 (int x, int y) {
  int z; ②
  z = 5 * func1 (x, y);
  return z;
}
main () {
   int a = 20; ①
   print (func2 (n, a)); ⑤
}
```

地点⑤実行時

変数	相対番地	実番地	値	変数	相対番地	実番地	値
~~n~~	~~-1,1~~	~~0~~	~~50~~			~~10~~	~~350~~
DL	0,-2	1	-1			11	50
RA	0,-1	2	-1			12	20
SL	0,0	3	-1			13	7
a	0,1	4	20			14	18
		5	50			15	9
		6	20			16	70
		7	1			17	-
		8	31			18	-
		9	3			19	-

FP 1 BP 3

89

オンライン試験

- 試験日 : 7月26 日(水)
- 試験時間 : 60分
- 試験範囲 : 第1～14回
- 配点 : 70点満点
- 持ち込み : 全て可
  - ただし外部との通信は禁止

90