平成26年度 3回生前期学生実験SW
最終課題（課題８）

1029-24-3152 竹田創

提出日：平成26 年8 月1 日

1　課題８

1.1　方針設計

kadai8.l は字句解析を行い、トークンとして返すデータを return で表す。
kadai8.y は主にパースを実行し、３つのセクションからなる。%token でトークンの宣言をする。ルールセクションで生成規則とアクションを記述する。C コードセクションで、タプルを構成する make_tuple とトークンを構成する make_token_node と定数ノードを表す make_constant_node の構造体を定義する。
kadai8_print.c は構文木の表示を行う
kadai8_semantic_analyser.c は意味解析を主に行う。
kadai8_code_generation.c はコード生成を主に行う。
yyerror() は構文エラーが生じたときにパーサが（自動的に）呼び出す関数である.
kadi7.h は構造体を定義し、データ生成関数のプロトタイプ宣言を行う。

1.2　各部の説明

1.2.1　kadai8_code_generation.c

主に kadai8_code_generation.c はコード生成を行う。その詳細な説明は次のセクションで説明するが、ここではその方針設計を説明する。実験資

料のように生成されるコード列を構造体のリストで表し、１命令のコード生成を行う関数としてコード生成を行う emit 関数を準備する。emit() の引数を一つ増やしてラベルも扱えるようにする。大域変数としてラベルの番号を記憶する変数を用意し、make_label 関数でラベルを作成する際に一意なラベルを作成する。また、make_return_label 関数ではリターンする際に使う一意なラベルを作成する。
大域変数を宣言するとき、大域データ領域にデータを割り当てる COMMON を使い、４バイトのメモリを割り当てる。関数定義のコードを生成するときに、top_alloc の値が 0 かどうかの判断をし 0 でなければ、つまり局所変数をもつとき関数本体のコードを生成する。
文のコード生成は while,if,if-else,return があり、分岐や条件分岐を用いる。それぞれの場合で分岐する可能性のある数のラベルを make_label 関数を使用して作り je や jmp でエミットする。
算術演算式のコードでは RSL 方式を用いた。関数を呼び出すときには、call する度に extern し、引数がいくつあるかを変数 parm_num を用いて計算し、esp に引数の４倍をたす。

1.2.2 kadai8.h

kadai8.h は構造体を定義する具体的には c(constant node),tk(token node),tp(4-tuple),nd(tree) の４つを構造体としてつくる。constant node 内で op はノードの種類、v は定数値を表す。token node では op はノードの種類、token 型の*next はポインタ、name はトークンの値、lev はオブジェクトが宣言されたブロックのレベル、kind はオブジェクトの種類、offset は相対番地（オブジェクトが局所変数及びパラメータの場合）やパラメータの数（関数の場合）を記憶を保持する整数値を表す。4-tuple 内で op はノードの種類を表し、a は枝を表すポインタである。
また、ヘッダを二重にインクルードしないようにインクルードガードを行う。

1.2.3 kadai8_semantic_analyser.c

yacc ファイルで変数宣言、パラメータ宣言、関数定義、変数参照、関数呼び出しの際に用いる解析用の関数を用意する。また、コード生成のときに必要となる、意味解析時のエラーをカウントする error 関数と警告を行う warn 関数を準備する。構造体 token によるスタックを操作する関数

として、lookup_sym、globalize_sym、pop_sym を定義する。lookup_sym はchar*型の引数をとり、同じ名前のオブジェクトがオブジェクト構造体のスタック上に存在するか調べる関数であり、lex ファイルで利用する。
globalize_sym は tree 型の引数で与えられるオブジェクト構造体を大域関数を表すオブジェクトとして登録し直す関数であり、関数呼び出しで利用する。
pop_sym は現在のブロックレベルを１減らすとき、そのスコープ内のオブジェクト構造体をスタックからポップする関数である。

1.2.4 kadai8.y

kadai8.y の宣言部でトークンの宣言をする。count_parm,offset,cur_lev を外部変数としてヘッダファイルをインクルードする前に宣言する。これにより他のファイルでもこの外部変数を利用することができる。TOKEN はchar*型、CONSTANT は int 型、IDENTIFIER と非終端記号は tree 型として宣言する。
ルールセクションでは生成規則とアクションを記述する。アクションで tree を返す時、枝が４未満の時は NULL を指定する。c 言語記述部では tree 型のトークンノード、定数ノード、タプルを作る関数を定義する。また構文木を表示する関数は print_program,print_external_declaration などで作成した。課題６からの変更点は意味解析を行う際に必要な変数や関数を組み込んだことである。

- 変数宣言を解析する make_decl 関数は declarator_list へ還元する際に使用する
- パラメータ宣言を解析する make_parm_decl 関数は parameter_declaration へ還元する際に使用する。
- 関数定義を解析する make_fun_def 関数は function_definition へ還元する際に使用する。
- 変数参照を解析する ref_var 関数は primary_expr へ還元する際に使用する。

- 関数参照を解析する ref_fun 関数は postfix_expr の IDENTIFIER へ還元する際に使用する。

また、現在のブロックレベルを保持する int 型の大域変数である cur_lev をパラメータリストの解析をはじめる時と複文の解析をはじめるときに１増やし、関数定義の解析を終えるときと複文の解析を終えるときに１減らす。関数のパラメータの数を記憶するため parameter_type_list が呼び出される度に int 型の変数 count_parm を１増やし関数定義の解析が終了したら declarator の offset に count_parm を保持しておく。そして関数を参照する際に argument_list を呼び出すたびに再び count_parmwo を１ずつ増やしていき、check_parm_num 関数でパラメータの数が正しいか判定する。

1.2.5 kadai8_print.c

1.3 実行方法

以下のようにして tinyc ファイルからアセンブリファイルを作成する。

```
a0129169@ws39:~/le3sw/kadai8/final_report$ bison -d kadai8.y &&  flex kadai8.l
a0129169@ws39:~/le3sw/kadai8/final_report$  ./tcc  < label.tc >label.asm1: warn
undeclared function

a0129169@ws39:~/le3sw/kadai8/final_report$ nasm -f elf label.asm && gcc -m32 -o

a0129169@ws39:~/le3sw/kadai8/final_report$ ./label
OK
```

1.4 実行例

例として上であげた label.asm のアセンブリコードを示す。

```
GLOBAL lab
lab: push ebp
mov ebp, esp
sub esp, 4
```

```
mov eax, 1
mov [ebp-4], eax
mov eax, 0
mov [ebp-8], eax
mov eax, [ebp-4]
cmp eax, 0
je L1
mov eax, [ebp-8]
cmp eax, 0
je L2
mov eax, 1
jmp R1
jmp L3
L2:
mov eax, 2
jmp R1
L3:
L1:
R1: mov esp, ebp
pop ebp
ret
GLOBAL main
main: push ebp
mov ebp, esp
sub esp, 4
mov eax, 2
push eax
call lab
add esp, 0
push eax
EXTERN chk
call chk
add esp, 8
R2: mov esp, ebp
pop ebp
```

```
ret
```

1.5 ソースコード

課題7から変更があったkadai8_semantic_analyser.cとkadai8_semantic_analyser.cのソースコードのみ示す。kadai8_semantic_analyser.cのソースコード

```
1  #include <stdio.h>
2
3  #include<string.h>
4  #include<stdarg.h>
5  #include <stdlib.h>
6  #include <assert.h>
7
8  #include "kadai8.h"
9  #include "kadai8.tab.h"
10
11
12 #include "kadai8_semantic_analyser.h"
13 #include "kadai8_print.h"
14 #include "kadai8_code_generation.h"講義資料2 1
15 //
16 int semnerrs;
17 int yylineno;
18
19 //はで使うparm_numcheck_parm_num
20 int parm_num;
21
22 int last_alloc ;
23 int top_alloc;教科書
24
25
26 //p.169
27 int allocate_loc(){
28   last_alloc=last_alloc -4;
29   if(last_alloc<top_alloc){
30     top_alloc=last_alloc;
31     //printf("allocate_loc/top_alloc:%d,last_alloc:%d",top_alloc,last_allo
32   }
33   return last_alloc;
34
35 }
```

```
36 /*
37   void release_one_loc(){
38   last_alloc += 4;
39   }*/
40 void release_loc(){
41   last_alloc+=4;
42
43 }
44 /*
45   int allocate_parm_loc(){
46   parm_alloc += 4;
47   return parm_alloc;
48   }*/
49
50 void error(char *fmt,...)
51 {
52   va_list argp;
53   va_start(argp, fmt);
54   semnerrs++;
55   fprintf(stderr, "%d: ", yylineno);
56   vfprintf(stderr, fmt, argp);
57   fprintf(stderr, "\n");
58   va_end(argp);
59 }
60 void warn(char *fmt,...)
61 {
62   va_list argp;
63   va_start(argp, fmt);
64   fprintf(stderr, "%d: warning: ", yylineno);
65   vfprintf(stderr, fmt, argp);
66   fprintf(stderr, "\n");
67   va_end(argp);
68 }講義資料２０　変数宣言を行う
69
70
71
72 //make_decl構文解析において，変数宣言部に対して次の関数を
73 //declarator-list へ還元するときのアクションとして実行する.
74
75 tree make_decl(tree n)
76 {
77   //  printf("make_decl\n");
78   switch (n->tk.kind) {
```

```
79    case VAR: // すでに変数として名前が宣言されている
80      /*p.19 変数宣言や関数定義によって新しいオブジェクトを作成すると
    き，同一レベルで同じ名前のオブジェクトがすでに存在していれば二重宣言二
    重定義エラーである
81        /
82
83      */
84
85      if (n->tk.lev == cur_lev)
86        // 同一レベルであれば二重宣言であるからエラー
87        {error("redeclaration of `%'s", n->tk.name);}そうで
    なければ，変数として新たなオブジェクトを作成し，
88      //kind をVAR にして，名前をそのオブジェクトに対応させる
89      allocate_loc();
90      n->tk.offset = last_alloc;
91      //printf("make_decl/tk.offset:%d",n->tk.offset);
92      //n = make_token_node(n->tk.name);
93      /*7/22    offset=offset - 4;
94        n->tk.offset=offset;*/
95      break;//してからをにしてbreakkindVARreturn
96
97
98
99    case FUN: // すでに関数として名前が定義されている
100   case UNDEFFUN: // すでに未定義関数として名前が使用されている
101     if (n->tk.lev == cur_levレベル（すなわち大域変数として宣言さ
    れている）ならば，二重宣言であるからエラー){//0
102       error`("%'s redeclared as different kind of symbol",n->tk.name);
103     }そうでなければ
104     //
105     n = make_token_node(n->tk.name);
106
107     break;//してからをにしてbreakkindVARreturn
108
109   case PARM: // すでにパラメータとして名前が宣言されている
110     warn("declaration of`%'s shadows a parameter", n->tk.name);
111     allocate_loc();
112     n->tk.offset = last_alloc;
113
114     //     n = make_token_node(n->tk.name);
115     /*7/22 offset=offset - 4;
116       n->tk.offset=offset;    */
117     break;
```

```
118
119     case FRESH (://kind がFRESH のとき), kind をVAR に変更するだ
    けでよい
120       //n = make_token_node(n->tk.name);
121       allocate_loc();
122       n->tk.offset = last_alloc;
123       /* 7/22 offset-=4;
124          n->tk.offset=offset;  */
125       break;
126     }
127     n->tk.kind = VAR;
128     //printf("make_decl/tk.offset:%d,tk.kinf:%d",n->tk.offset ,n->tk.kind);
129     return n;
130   }パラメータ宣言
131
132   //
133   tree make_parm_decl(tree n)
134   {//printf("make_parm_decl");
135     switch (n->tk.kind) {
136     case VAR:
137       n = make_token_node(n->tk.name);
138       offset= offset+4;
139       n->tk.offset=offset;
140       break;
141     case FUN:
142     case UNDEFFUN:
143       n = make_token_node(n->tk.name);
144       break;
145     case PARM二重宣言がおこるのはパラメータ同士の場合に限られる://
146       error("redeclaration of `%'s", n->tk.name);
147       return n;
148     case FRESH:
149       // n = make_token_node(n->tk.name);
150       offset=offset+4;
151       n->tk.offset=offset;
152       break;
153     }
154     n->tk.kind = PARM;変更
155     //7/22
156     //n->tk.offset = allocate_parm_loc();
157     //  n->tk.offset=offset_parm();
158     //printf("tk.offset:%d",n->tk.offset);
159     return n;
```

```
160 }関数定義
161
162
163
164 //
165
166
167 tree make_fun_def(tree n)
168 {//printf("make_fun_def");
169   switch (n->tk.kind) {
170   case VAR関数同士または大域変数宣言との間であるとき二重宣言://
171     error‘ ("% 's redeclared as different kind of symbol",
172           n->tk.name);
173     break;
174   case FUN:
175     error("redefinition of ‘ % 's", n->tk.name);
176     break;
177   case UNDEFFUN:
178     if(n->tk.offset != parm_num)
179       error("function '%s'; the number of parameter is wrong", n->tk.name)
180
181   case FRESH:
182     // printf(" n->tk.kind = FUN,%s;",n->tk.name);
183
184     n->tk.kind = FUN;
185     break;
186   case PARM:
187     break;
188
189   }変更
190   //7/22
191   /*n->tk.offset = parm_num;
192     parm_num = 0;
193     return n;*/
194   return n;
195 }変数参照
196
197 //
198 tree ref_var(tree n){
199   //printf("ref_var\n");
200   switch (n->tk.kind) {
201   case VAR:
202   case PARM:
```

```
203       break;
204     case FUN:
205     case UNDEFFUN:
206       error("function `%s'(n->tk.name) is used as variable(ref_var)", n->tk.n
207       break;
208     case FRESH:
209       error`("%s'(n->tk.name) undeclared variable(ref_var)", n->tk.name);
210       n->tk.kind = VAR; /* エラーリカバリ*/
211       break;
212     }
213     return n;
214 }関数呼び出し
215 //
216 tree ref_fun(tree n)
217 {//printf("ref_fun%s\n",n->tk.name);
218     switch (n->tk.kind) {
219     case VAR:
220     case PARM:
221       error("variable `%s' is used as function", n->tk.name);
222       break;
223     case FUN:
224     case UNDEFFUN:
225
226       break;
227     case FRESH:
228       warn` ("% 's undeclared function", n->tk.name);
229       n->tk.kind = UNDEFFUN;
230       if (n->tk.lev > 0) globalize_sym(n);
231       else //printf("ref_fun(): unglobalized_sym()\n");
232       break;
233     }
234     return n;
235 }
236
237
238 // 変更
239
240 void check_parm_num(int c, tree n) {
241           if(n->tk.kind != UNDEFFUN && n->tk.offset < c) error("too many arg
242           if(n->tk.kind != UNDEFFUN && n->tk.offset > c) error("too few argu
243 }
244
245
```

```
246 int count_argument_expression(tree n){
247   if(n==NULL){
248     return 0;
249   }
250   if(n->tp.op==CONS){
251     return   1+count_argument_expression(n->tp.a[1]);
252   }else{
253     return 1;
254   }
255 }
256
257
258
259
260 //p.19 pop?
261 void pop_sym(){スコープを終えるレベル値を持つ構造体をたどり，たどり
    終えた構造体の次の構造体を
262
263   //symtab で指すようにすればよい
264
265   if(symtab==NULL){
266
267     error("symrtab must not be null\n");
268   }
269   while(symtab->lev > cur_lev){
270     // printf("pop_sym{symtab->lev: %d, cur_lev:%d, symtab->name:%s}\n", s
271     symtab =(token)symtab->next;
272   }
273
274
275
276
277
278 }
279 void globalize_sym(tree n引数で与えられるオブジェクト構造体を大域
    関数を表すオブジェクトとして登録し直す関数)//
280 //n->tk.がであるがこの関数にわたされるkindFRESHn 1
281 //にさされているポインタをの次のポインタをさすようにするUNDEFUNDEF
282 //2 最初のオブジェクト（が）のの先にをもっていくnextNULLnextUNDEF
283 {
284   token findout_undeffun = symtab;
285   token findout_undeffun_before = NULL;
286   token looking = symtab;
```

```
287
288
289     //のトークンを探すundeffun
290     while(findout_undeffun != &n->tk){//はのオブジェクトであり、
    みつけるまで走査nundef
291       findout_undeffun_before = findout_undeffun;
292       findout_undeffun = findout_undeffun->next;
293     }
294
295     assert(strcmp(findout_undeffun->name, n->tk.name) == 0);
296     //printf("GS:%s\n", n->tk.name);
297
298     //を抜く分の修正undeffun
299     if(findout_undeffun_before)
300       findout_undeffun_before->next = findout_undeffun->next;//
    があればをとびこしてつなぐbeforeundeffun
301     else{
302       symtab = findout_undeffun->next;
303       looking = symtab;
304     }
305
306
307     //をレベル0にして最初にもってくるfindout_undeffun
308     findout_undeffun->lev = 0;
309     findout_undeffun->next = NULL;
310
311
312     //のトークンを探す。symtab
313     while(looking->next){
314       looking = looking->next;
315     }挿入する
316     //
317     looking->next = findout_undeffun;
318
319
320
321 }同じ名前のオブジェクトがオブジェクト構造体のスタック上に存在するか調べ
    る関数
322
323
324
325 //
326 //.で使用するl
```

```
327 tree lookup_sym(char *yytext){
328   //printf("lookup_sym\n");名前を引数にとり、スタックを上から順に
    走査して同名のオブジェクトがみつかればそれを
329   //型の値として返すtree
330
331   //token tmp=symtab;デバッグ用
332   /*
333     printf("lookup[%s]:", yytext);
334     while(tmp!=NULL){
335     printf("/%s:%d",tmp->name, tmp->lev);
336     tmp=tmp->next;
337     }
338     printf("\n");
339   */
340   token looking_stack=symtab;
341
342
343   while(looking_stack!=NULL最後のオブジェクトでなければ繰り返す
    ){//(の初期値はsymtabNULL)
344
345     if(strcmp(looking_stack->name,yytext同名のオブジェクトがあ
    ればそれを)==0){//型の値として返すtree
346       //    printf("hit/%s:%d\n",looking_stack->name, looking_stack->lev);
347       return (tree)looking_stack;
348     }else次のスタックを走査して{//に戻るwhile
349       //printf("lookup_sym{symtab->lev: %d, cur_lev:%d, symtab->name:%s}\n
350       looking_stack=looking_stack->next;
351     }最後まで同名のオブジェクトがみつからなければ
352   }//を返すNULL(の初期値はsymtabNULL)
353   return NULL  ;
354 }
355
356
357 tree lookup_num(int num) {
358   tree  t =(tree) symtab;
359   while(t){
360     if(num >= t->tk.lev) return t;
361     else t = (tree)t->tk.next;
362     offset = offset + 4;
363   }
364   return t;
365 }
366
```

```
367
368
369
370 //呼び出されたということはスタックに積むことがきまった事後make_token_node
371 tree make_token_node(char *str){
372   //ののにをいれるtreetokennameidentifier新しく生成された構造体を
    スタックに積む（
373
374
375   //。１８）pスタックにむ。
376   //tree_p->をいじくるnext構造体のメモリ割り当て
377
378
379   //
380   tree tree_p=(tree) malloc (sizeof(*tree_p));
381
382   tree_p->tk.op =TOKEN;
383   tree_p->tk.name= strdup(str);//のとするstrmalloccopystrdup
384
385   tree_p->tk.lev=cur_lev現在のブロックレベルを保持する;//int 型
    の大域変数cur_lev を導入しておき，make_token_nodeでオブジェクト構
    造体を生成するときに，()cur_lev の値をlev の値として記憶する（p）.18
386   // printf("FUGAFUGA %s : %d\n", tree_p->tk.name, tree_p->tk.lev);
387   tree_p->tk.kind=FRESH;
388   tree_p->tk.next=symtab;//symtab はtoken 型の大域変数であり，
    リストの先頭すなわちスタックのトップを指すのに使用する.
389   // tree_p->tk.offset=offset_関数();//はで定義するoffsetsemantic_analyser
390   symtab= (token)tree_p;//の初期値はsymtabNULL
391   return tree_p;
392 }
393
394 tree make_constant_node(int op){
395   //tree構造体のメモリ割り当て
396
397   //
398   //printf("make_constant_node\n");
399   tree tree_p=(tree) malloc (sizeof(*tree_p));
400   tree_p->c.op =CONSTANT;
401   tree_p->c.v =op;
402   //   printf("make_constant_node,op:%d\n",op);
403   return tree_p;
404 }
405
```

```
406 tree make_tuple(int op,tree a0, tree a1,tree a2, tree a3 ){
407   //  printf("make_tuple\n");構造体のメモリ割り当て
408   //
409   tree tree_p=(tree) malloc (sizeof(*tree_p));
410
411
412   tree_p->tp.op=opノードの種類を;//のノードに入れるtree
413   tree_p->tp.a[0] =a0;
414   tree_p->tp.a[1] =a1;
415   tree_p->tp.a[2] =a2;
416   tree_p->tp.a[3] =a3;
417   //printf("make_tuple,op:%d\n",op);
418   return tree_p;
419 }
```

kadai8_code_generation.c のソースコード

```
 1
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <string.h>
 5
 6 #include "kadai8.h"
 7 #include "kadai8.tab.h"
 8 #include "kadai8_semantic_analyser.h"
 9 #include "kadai8_print.h"
10 #include "kadai8_code_generation.h"
11
12
13 struct code *final_code;
14 struct code *first_code;
15 char *label_return_jump;
16 int label_count=0;
17 int return_label_count=0;
18 //int parm_num=0;
19 int parm_num_esp;
20
21 //で宣言したのになぜ必要？yacc資料7．2
22
23
24 //命令のコード生成を行う関数が
25 //1emit
26 //の引数を一つふやしてラベルも扱うemit
27 struct code *emit(char *inst, char *op1, char *op2, char *label){
```

```
28    char buf[80];
29    struct code *c = (struct code *)malloc(sizeof(struct code));
30
31
32    if (label == NULL資料){//
33      if (inst == NULL)
34        buf[0] = '\0';
35      else if (op1 == NULL)//はここにいれるcommon?
36        sprintf(buf, "\t%s\n", inst);
37      else if (op2 == NULL)
38        sprintf(buf, "\t%s\t%s\n", inst, op1);
39      else
40        sprintf(buf, "\t%s\t%s, %s\n", inst, op1, op2);
41    }else{//label!=のときをそしての先頭にいれるNULLlabelbufc
42      if (inst == NULL)
43        sprintf(buf, "%s\n", label);
44      else if (op1 == NULL)
45        sprintf(buf, "%s\t%s\n", label, inst);
46      else if (op2 == NULL)
47        sprintf(buf, "%s\t%s\t%s\n", label, inst, op1);
48      else
49        sprintf(buf, "%s\t%s\t%s, %s\n", label, inst, op1, op2);
50    }
51
52    c->cd = strdup(buf);
53    c->next=NULL;資料
54    //p.28 構造体をコード列リストの最後に追加c
55
56    if(first_code){//コードがまだ定義されているときfirst
57      final_code->next=c最終尾につなぎ、つないだものを最終尾とする
    ;//
58      final_code=c;
59    }else{ //first ,コードに最初に定義するときfinal
60      first_code=c;
61      final_code=c;
62    }
63    return c;
64  }一意的なラベルを生成。整数型の大域変数
65  //でラベルのカウント。にたいしてを生成label_countnLabel_N(p.29)
66  char *make_label(){
67    char label_count_string[64];
68    char L[64] = "L";
69    char *Label_N;
```

```
70
71    label_count++;
72
73    //型のをに変換intlabel_countchar
74    sprintf(label_count_string , "L%d", label_count );
75
76    //   strcat(L, label_count_string文字列配列);//の後ろにを連結
      Llabel_count_char(ではエラーlabel_count)
77    Label_N = strdup(label_count_string );//とを行いポインタを代
      入malloccopy
78    // printf("make_counrLabel_N:%s",Label_N );
79
80    return Label_N;
81  }
82
83
84
85  //"L"+"return_label_count"+"retを返す"
86  char *make_label_return(){
87    char label_count_char[64];
88    char L[64] = "R";
89    char *Label_N_ret;
90
91    return_label_count++;
92
93    //型のをに変換intlabel_countchar
94    //printf("make_label_return\n");
95    sprintf(label_count_char , "%d", return_label_count );
96
97    strcat(L, label_count_char文字列配列);//の後ろにを連結Llabel_count_char(
      ではエラーlabel_count)
98    Label_N_ret = strdup(L);//とを行いポインタを代入malloccopy
99
100   return Label_N_ret;
101 }
102
103
104
105
106
107
108 char *ebp_set(tree p) {
109   char *ebp_buf = (malloc(sizeof(char) *4 ));
```

```
110   if(p->tk.lev == 0) sprintf(ebp_buf, "[%s]", p->tk.name);
111   else if(p->tk.offset > 0) sprintf(ebp_buf, "[ebp+%d]", p->tk.offset);
112   else if(p->tk.offset < 0) sprintf(ebp_buf, "[ebp%d]", p->tk.offset);
113   return ebp_buf;
114 }
115
116
117 //yacc でつかわれる満数でコード列リストを表示する
118 void print_code(){
119   struct code *now_printing=first_code;
120   while(now_printing){
121
122     printf("%s",now_printing->cd);
123     now_printing=now_printing->next;
124   }
125 }
126
127
128
129 //1
130 void emit_program(tree p) {
131   top_alloc = 0;
132   last_alloc = 0;
133
134   // printf("1program\n");
135   if (p->n.op != CONS) {
136     //external_declaration {$$=$1;}
137     emit_external_declaration(p);
138
139   } else {
140     //|program external_declaration {$$=make_tuple(CONS,$1,$2,NULL,NULL);}
141     emit_program(p->tp.a[0]);
142     emit_external_declaration(p->tp.a[1]);
143
144   }
145 }
146 //2
147 void emit_external_declaration(tree p) {
148   //printf("2external_declaration\n");
149   if (p->n.op == DECLARATION) {
150     // printf("(");
151     emit_declaration(p);
152     // printf(")\n");
```

```
153     }else if(p->n.op ==FUNDEF1){
154       //printf("(");
155       emit_function_definition(p);
156       //printf(")\n");
157     }
158   }
159
160
161   //3
162   void emit_declaration(tree p){
163     // printf("3declaration\n");
164     //  printf("(");
165     //printf("INT");
166
167     emit_declarator_list(p->tp.a[0]);
168     //printf(";");
169     //printf(")\n");
170
171
172   }
173
174   //4
175   void emit_declarator_list(tree p){
176     //printf("4declarator_list\n");
177     if (p->n.op ==CONS) {
178       emit_declarator_list(p->tp.a[0]);
179       //printf(",");
180       emit_declarator(p->tp.a[1]);
181
182     }else{
183       emit_declarator(p);
184     }
185   }
186
187   //5
188   void emit_declarator(tree p){
189     // printf("5declarator\n");
190     /* printf("IDENTIFIER "); */大域変数の定義するとき、大域データ
      領域にデータを割り当てる
191
192     //を使うcommonバイトのメモリを大域データ領域に割当て、そのアドレスを
193     //4とするlabel資料(p.27)
194     if(p->tk.lev == 0){
```

```
195       char common_label_4[64];
196       char str[100];
197       strcpy(common_label_4, p->tk.name);
198
199       sprintf(str,"COMMON\t%s\t4\n", common_label_4);
200
201       //関数を使って表示するemit
202       emit(str,NULL,NULL,NULL);
203     }
204
205
206   }
207
208
209   //6 資料関数定義のコード生成 7.2
210   void emit_function_definition(tree p){
211     // printf("6function_definition\n");
212     if(p->n.op==FUNDEF1){
213       //int declarator:a[0](parameter_type_list_opt:a[1]){compound_statement
214
215       //  printf("¥INTn");
216       char *function_name;
217       struct code *c;
218       char label_function_name[64];
219       char label_return[64];
220       function_name = strdup(p->tp.a[0]->tk.name);
221       label_return_jump = make_label_return();
222       strcpy(label_return, label_return_jump);
223       strcat(label_return, ":");
224       strcpy(label_function_name, function_name);
225       strcat(label_function_name, ":");関数定義のコード生成
226
227       // p.29
228       emit("GLOBAL", function_name, NULL, NULL);
229       emit("push", "ebp", NULL, label_function_name);
230       emit("mov", "ebp", "esp", NULL);
231       c = emit(NULL, NULL, NULL, NULL);関数本体のコード生成
232       // p.29
233       emit_parameter_type_list_opt(p->tp.a[1]);
234       emit_compound_statement(p->tp.a[2]);
235       //の部分のコード生成Nlocal p.29
236       //は最後にわりあてられまだ開放されていない局所変数の相対番地last_alloc
```

```
237       //はこれまでのの最小値でtop_alloctop_alloc関数本体のコード生
      成が終了したとき
238       //を符号反転させたものがとなる（教科書top_allocNlocalp）.168
239       // printf("top_alloc!=0:%d",top_alloc);
240       if (top_alloc!=0){//がでなければNlocal0
241
242         char buf[80];
243         sprintf(buf, "\tsub\tesp, %d\n", -top_alloc);
244         c->cd = strdup(buf);
245         emit("mov", "esp", "ebp", label_return);
246         emit("pop", "ebp", NULL, NULL);
247         emit("ret", NULL, NULL, NULL);
248       } else {//がならばNlocal0
249         emit("pop", "ebp", NULL, label_return);
250         emit("ret", NULL, NULL, NULL);
251       }
252
253     }
254
255
256   }
257
258   //6-2
259   void emit_parameter_type_list_opt(tree p){
260     if(p==NULL){}
261     else{
262       emit_parameter_type_list(p);
263     }
264
265   }
266   //7
267   void emit_parameter_type_list(tree p){
268     //printf("7parameter_type_list\n");
269     if(p->n.op!= CONS){
270       emit_parameter_declaration(p);
271
272     }else{
273       emit_parameter_type_list(p->tp.a[0]);
274       //   printf(",");
275       emit_parameter_declaration(p->tp.a[1]);
276
277     }
278   }
```

```
279
280
281 //8
282 void emit_parameter_declaration(tree p){
283   // printf("8parameter_declaration\n");
284   //printf("¥INTn");
285   //emit_parm(p);
286   emit_declarator(p);
287 }
288 //9
289 void emit_statement(tree p){
290   // printf("9statement\n");
291   if(p==NULL){
292     //printf(";\n");
293     return;
294
295   }
296   switch(p->tp.op){
297   case STATEMENT_EXPRESSION: {
298     //   if(p->n.op== STATEMENT_EXPRESSION){
299     emit_expression(p->tp.a[0]);
300     // printf(";");
301     // printf("\n");
302
303
304
305     break;
306   }
307   case STATEMENT_IF:{ラベル生成
308
309
310
311     //
312     char *label_if_jump = make_label();
313
314     char label_if[64];
315     strcpy(label_if, label_if_jump);
316
317     strcat(label_if, ":");コード生成
318
319     //条件判定
320     //
321     emit_expression(p->tp.a[0]);
```

```
322
323       emit("cmp", "eax", "0", NULL);
324       emit("je", label_if_jump, NULL, NULL);
325       //だった場合の処理true
326       emit_statement(p->tp.a[1]);
327       emit(NULL, NULL, NULL, label_if);
328
329       break;
330     }
331     case STATEMENT_IF_ELSE:{教科書
332
333       //p.173ラベル生成
334       //
335       char *label_false_jump = make_label();
336       char *label_right_jump = make_label();
337       char label_false[64], label_right[64];
338       strcpy(label_false, label_false_jump);
339       strcpy(label_right, label_right_jump);
340       strcat(label_false, ":");
341       strcat(label_right, ":");コード生成
342       //条件判定
343       //
344       emit_expression(p->tp.a[0]);
345
346       emit("cmp", "eax", "0", NULL);
347       emit("je", label_false_jump, NULL, NULL);
348
349       emit_statement(p->tp.a[1]);
350       emit("jmp", label_right_jump, NULL, NULL);
351       emit(NULL, NULL, NULL, label_false);
352       emit_statement(p->tp.a[2]);
353       emit(NULL, NULL, NULL, label_right);
354
355       break;
356     }
357     case STATEMENT_WHILE:{
358       // }else if(p->n.op== STATEMENT_WHILE){資料
359       //p.30 教科書p.174先頭ラベル作成
360       //
361       char *label_first_jump = make_label();末尾ラベル作成
362       //
363       char *label_last_jump = make_label();
364       char label_first[64], label_last[64];
```

```
365       strcpy(label_first , label_first_jump);
366       strcpy(label_last , label_last_jump);
367       strcat(label_first , ":");
368       strcat(label_last , ":");条件式のコード生成
369       //
370       emit(NULL, NULL, NULL, label_first);
371       emit_expression(p->tp.a[0]);
372       emit("cmp", "eax", "0", NULL);
373       emit("je", label_last_jump , NULL, NULL);
374       //true本体のコード生成:
375       emit_statement(p->tp.a[1]);
376       emit("jmp", label_first_jump , NULL, NULL);
377       //false末尾ラベル挿入:
378       emit(NULL, NULL, NULL, label_last);
379
380       break;
381     }
382     case STATEMENT_RETURN:{
383       //   }else if(p->n.op== STATEMENT_RETURN){
384
385       // printf("¥RETURNn");
386
387       if (p->tp.a[0] != NULL)
388         emit_expression(p->tp.a[0]);
389       //で定義したラベルに飛ぶFUNDEFreturn
390       emit("jmp", label_return_jump , NULL, NULL);
391
392
393
394
395       break;
396     }
397     default:
398       // printf("emit_compound_statement");
399       emit_compound_statement(p);
400       break;
401     }
402   }
403
404   //10
405   void emit_compound_statement(tree p){
406     //printf("10compound_statement\n");
407     if(p->n.op== COMP_STATE){
```

```
408       //printf("{");
409       // if(p->tp.a[0]!=NULL){
410       // printf("PIYOPIYO\n");
411       // printf("opt_decl !=NULL\n");
412       emit_opt_declaration_list(p->tp.a[0]);
413       // }else{
414       // printf("opt_decl =NULL\n");
415       // }
416       // if (p->tp.a[1]!=NULL){
417       // printf("opt_state!=NULL\n");
418
419       emit_opt_statement_list(p->tp.a[1]);
420       // }else{
421       // printf("opt_state =NULL\n");
422       // }
423       // printf("}\n");
424
425       // }else{
426       //  printf("error:compound statement");
427       //}
428     }
429   }
430   //10-2
431   void emit_opt_declaration_list(tree p){
432
433     //printf("10-1opt_declaration_list\n");
434     // printf("%d\n",p->tp.a[0]) ;
435
436     if(p!=NULL){
437       //printf("!=NULL  opt_declaration_list\n");
438       emit_declaration_list(p);
439     }else{
440       // printf("NULL   opt_declaration_list\n");}
441     }
442   }
443   //10-3
444   void emit_opt_statement_list(tree p){
445     // printf("opt_statement_list\n");
446     if(p!=NULL){
447       // printf("!=NULL opt_statement_list\n");
448       emit_statement_list(p);
449     }else{
450       // printf("NULL opt_statement_list\n");
```

```
451       //emit_statement_list(p/\*->tp.a[1]*\/);
452
453     }
454   }
455
456
457
458   //11
459   void emit_declaration_list(tree p){
460     // printf("11 declaration_list\n");
461     if(p->n.op!= CONS){
462       emit_declaration(p);
463       allocate_loc();
464
465     }else{
466       emit_declaration_list(p->tp.a[0]);
467
468       emit_declaration(p->tp.a[1]);
469       allocate_loc();
470
471     }
472   }
473   //12
474   void emit_statement_list(tree p){
475     // printf("12 statement_list\n");
476     if(p->n.op!= CONS){
477       // printf("!=cons\n");
478       emit_statement(p);
479
480     }else{//printf("else");
481       emit_statement_list(p->tp.a[0]);
482       emit_statement(p->tp.a[1]);
483
484     }
485   }
486   //13
487   void emit_expression(tree p){
488     // printf("13 expression\n");
489     if(p->n.op!= CONS){
490       emit_assign_expr (p);
491
492     }else{
493       emit_expression (p->tp.a[0]);
```

```
494       // printf(", ");
495       emit_assign_expr (p->tp.a[1]);
496
497     }
498   }
499   //14
500   void emit_assign_expr(tree p){
501
502     // printf("14assign_expr\n");
503     if(p->n.op==ASSIGN教科書){//p.176コード作成
504
505       //
506       emit_assign_expr(p->tp.a[1]);追加)
507       //(7/25
508       char *identifier = (malloc(sizeof(char) *4 ));
509       identifier = ebp_set(p->tp.a[0]);
510
511
512       emit("mov", identifier , "eax", NULL);
513
514     } else{
515       emit_logical_OR_expr(p);
516
517     }
518   }
519
520   //15
521   void emit_logical_OR_expr(tree p){教科書
522     //p.193
523     // printf("15logical_OR_expr\n");
524     if(p->n.op!= OR){
525       emit_logical_AND_expr(p);
526
527     }else{準備一時変数を用意、ラベル生成
528
529
530       //()
531       char temp[64] = "[ebp", address[64];
532       char d_word[64] = "dword ";
533       char *label_j = make_label();
534       char label[64];
535       strcpy(label , label_j);
536       strcat(label , ":");
```

```
537       sprintf(address, "%d", allocate_loc());
538       strcat(temp, address);
539       strcat(temp, "]");
540       strcat(d_word, temp);コード生成
541
542       //
543       emit("mov", d_word, "1", NULL);
544       emit_logical_OR_expr(p->tp.a[0]);
545       emit("cmp", "eax", "1", NULL);
546       emit("je", label_j, NULL, NULL);
547       emit_logical_AND_expr(p->tp.a[1]);
548       emit("cmp", "eax", "1", NULL);
549       emit("je", label_j, NULL, NULL);
550       emit("mov", d_word, "0", NULL);
551       emit("mov", "eax", temp, label);
552
553       release_loc();
554     }
555
556
557 }
558 //16
559 void emit_logical_AND_expr(tree p){教科書
560   //p.193
561   //printf("16logical_AND_expr\n");
562   if(p->n.op!= AND){
563     emit_equality_expr(p);
564   }else{準備一時変数を用意、ラベル生成
565     //()
566     char temp[64] = "[ebp", address[64];
567     char d_word[64] = "dword ";
568     char *label_j = make_label();
569     char label[64];
570     strcpy(label, label_j);
571     strcat(label, ":");
572     sprintf(address, "%d", allocate_loc());
573     strcat(temp, address);
574     strcat(temp, "]");
575     strcat(d_word, temp);コード生成
576
577     // 資料p.32 e1&&e2
578     //mov dword temp,0
579     emit("mov", d_word, "0", NULL);
```

```
580       emit_logical_AND_expr(p->tp.a[0]);
581       //が偽ならへe1label_j
582       emit("cmp", "eax", "0", NULL);
583       emit("je", label_j, NULL, NULL);
584       //が偽ならへe2label_j
585       emit_equality_expr(p->tp.a[1]);
586       emit("cmp", "eax", "0", NULL);
587       emit("je", label_j, NULL, NULL);
588       //mov dword temp,1
589       emit("mov", d_word, "1", NULL);
590       //label_j:mov eax temp
591       emit("mov", "eax", temp, label);
592
593       release_loc();
594     }
595
596
597 }
598
599
600 //17
601 void emit_equality_expr(tree p){
602   //printf("n.op:%d\n",p->n.op);
603   // printf("17equality_expr\n");
604
605   if(p->n.op==NOT_EQUAL_TO ){
606     //  printf("not equal_to");一時変数の用意
607
608
609       //
610       char temp[64] = "[ebp", address[64];
611       sprintf(address, "%d", allocate_loc());
612       strcat(temp, address);
613       strcat(temp, "]");コード生成資料
614
615       //(p.31 e1!=e2)
616       //の計算e2
617       emit_equality_expr(p->tp.a変更[0]);//
618       //mov temp eax
619       emit("mov", temp, "eax", NULL);
620       //の計算e1
621       emit_relational_expr(p->tp.a変更[1]);//
622       //cmp eax,temp
```

```
623         emit("cmp", "eax", temp, NULL);
624
625
626
627
628
629
630
631         //setne al
632         emit("setne", "al", NULL, NULL);
633         //movzx eax al
634         emit("movzx", "eax", "al", NULL);
635
636         release_loc();
637
638
639
640     }else if(p->n.op==EQUAL_TO ){
641         //printf("equal to");一時変数を用意
642
643
644         //
645         char temp[64] = "[ebp", address[64];
646         sprintf(address, "%d", allocate_loc());
647         strcat(temp, address);
648         strcat(temp, "]");コード生成資料
649
650         //(p.31 e1==e2)
651         //の計算e2
652         emit_equality_expr(p->tp.a[0]);
653         //mov temp,eax
654         emit("mov", temp, "eax", NULL);
655         //の計算e1
656         emit_relational_expr(p->tp.a[1]);
657         //cmp eax,temp
658         emit("cmp", "eax", temp, NULL);
659         //sete al
660         emit("sete", "al", NULL, NULL);
661         //movzx eax al
662         emit("movzx", "eax", "al", NULL);
663
664         release_loc();
665
```

```
666     }else{
667       // printf("else");
668
669
670
671       emit_relational_expr(p);
672
673
674     }
675   }
676   //18
677   void emit_relational_expr(tree p){
678     //printf("18relational_expr");
679     if(p->n.op==GREATER_THAN){一時変数を用意
680
681       //
682       char temp[64] = "[ebp", address[64];
683       sprintf(address, "%d", allocate_loc());
684       strcat(temp, address);
685       strcat(temp, "]");コード生成
686
687       // 資料(p.31 e1!=e2)
688       //の計算e2
689
690       emit_relational_expr(p->tp.a変更[0]);//
691       //mov temp eax
692       emit("mov", temp, "eax", NULL);
693       //の計算e1
694       emit_add_expr(p->tp.a変更[1]);//
695       //cmp eax,temp
696       emit("cmp", "eax", temp, NULL);
697       //setl al
698       emit("setl", "al", NULL, NULL);
699       //movzx eax al
700       emit("movzx", "eax", "al", NULL);
701
702       release_loc();
703
704
705     }else if(p->n.op==LESS_THAN){一時変数を用意
706       //
707       char temp[64] = "[ebp", address[64];
708       sprintf(address, "%d", allocate_loc());
```

```
709         strcat(temp, address);
710         strcat(temp, "]");コード生成
711         //
712         emit_relational_expr(p->tp.a変更[0]);//
713         //  emit_add_expr(p->tp.a[1]);
714         emit("mov", temp, "eax", NULL);
715         emit_add_expr(p->tp.a変更[1]);//
716         // emit_relational_expr(p->tp.a[0]);
717         emit("cmp", "eax", temp, NULL);
718         emit("setg", "al", NULL, NULL);
719         emit("movzx", "eax", "al", NULL);
720
721         release_loc();
722
723     }else if(p->n.op==GREATER_THAN_EQUAL){一時変数を用意
724
725         //
726         char temp[64] = "[ebp", address[64];
727         sprintf(address, "%d", allocate_loc());
728         strcat(temp, address);
729         strcat(temp, "]");コード生成
730
731         //
732         emit_relational_expr(p->tp.a[0]);
733
734         //   emit_add_expr(p->tp.a[1]);
735         emit("mov", temp, "eax", NULL);
736         emit_add_expr(p->tp.a[1]);
737         //       emit_relational_expr(p->tp.a[0]);
738         emit("cmp", "eax", temp, NULL);
739         emit("setle", "al", NULL, NULL);
740         emit("movzx", "eax", "al", NULL);
741
742         release_loc();
743
744     }else if(p->n.op== LESS_THAN_EQUAL){一時変数を用意
745
746         //
747         char temp[64] = "[ebp", address[64];
748         sprintf(address, "%d", allocate_loc());
749         strcat(temp, address);
750         strcat(temp, "]");コード生成
751
```

```
752       //
753       emit_relational_expr(p->tp.a[0]);
754       // emit_add_expr(p->tp.a[1]);
755       emit("mov", temp, "eax", NULL);
756       //emit_relational_expr(p->tp.a[0]);
757       emit_add_expr(p->tp.a[1]);
758       emit("cmp", "eax", temp, NULL);
759       emit("setge", "al", NULL, NULL);
760       emit("movzx", "eax", "al", NULL);
761
762       release_loc();
763
764
765     }else{
766       emit_add_expr(p);
767     }
768   }
769
770   //19
771   void emit_add_expr(tree p){
772     // printf("19add_expr");
773     if(p->n.op== ADD){一時変数を用意
774
775       //
776       char temp[64] = "[ebp", address[64];
777       sprintf(address, "%d", allocate_loc());
778       strcat(temp, address);
779       strcat(temp, "]");コード生成
780
781       //
782       emit_mult_expr(p->tp.a[1]);
783       //   emit_add_expr(p->tp.a[0]);右オペランドのコード生成
784       //
785       //emit("mov一時変数",,"eax");
786       emit("mov", temp, "eax", NULL);
787       emit_add_expr(p->tp.a[0]);
788       //     emit_mult_expr(p->tp.a[1]);左オペランドのコード生成
789       //
790       //emit命令("","eax一時変数",);
791       emit("add", "eax", temp, NULL);一時変数の解放
792       //
793       release_loc();
794
```

```
795     } else if(p->n.op== SUB){引き算のマイナス
796
797
798       //一時変数を用意
799       //
800       char temp[64] = "[ebp", address[64];
801       sprintf(address, "%d", allocate_loc());
802       strcat(temp, address);
803       strcat(temp, "]");コード生成
804
805       //
806       // emit_add_expr(p->tp.a[1]);
807       emit_mult_expr(p->tp.a[1]);
808
809       emit("mov", temp, "eax", NULL);引き算
810       //
811       emit_add_expr(p->tp.a[0]);
812       //   emit_mult_expr(p->tp.a[0]);
813
814       emit("sub", "eax", temp, NULL);
815       release_loc();
816
817
818     }else{
819       emit_mult_expr(p);}
820
821 }
822
823 //20
824 void emit_mult_expr(tree p){
825   // printf("20mult_expr");
826   if(p!=NULL){
827
828
829
830
831       if(p->n.op== MULTI){一時変数を用意
832
833         //
834         char temp[64] = "[ebp", address[64];
835         sprintf(address, "%d", allocate_loc());
836         strcat(temp, address);
837         strcat(temp, "]");コード生成
```

```
838
839         //
840         //    emit_unary_expr(p->tp.a[1]);
841         emit_mult_expr(p->tp.a[0]);
842         //     emit_mult_expr(p->tp.a[0]);
843         emit("mov", temp, "eax", NULL);
844         // emit_mult_expr(p->tp.a[0]);
845         emit_unary_expr(p->tp.a[1]);
846         emit("imul", "eax", temp, NULL);
847
848         release_loc();
849
850       }else if(p->n.op== DIV){資料
851         //p.31 e1/e2一時変数を用意
852         //
853         char temp[64] = "[ebp", address[64];
854         char d_word[64] = "dword ";
855         sprintf(address, "%d", allocate_loc());
856         strcat(temp, address);
857         strcat(temp, "]");
858         strcat(d_word, temp);コード生成
859
860         //
861         //の計算e2
862         //   emit_unary_expr(p->tp.a[0]);
863         emit_mult_expr(p->tp.a[0]);
864         //mov temp,eax
865         emit("mov", temp, "eax", NULL);
866         //の計算e1
867         emit_unary_expr(p->tp.a[1]);
868         // emit_mult_expr(p->tp.a[1]);
869         //cdq
870         emit("cdq", NULL, NULL, NULL);
871         //idiv dword temp
872         emit("idiv", d_word, NULL, NULL);
873
874         release_loc();
875
876       }else{
877         emit_unary_expr(p);
878       }
879     }else {
880       // printf("p=NULL");
```

```
881     }
882   }
883   //21
884   void emit_postfix_expr(tree p){
885     // printf("21postfix_expr");教科書
886     //p.160 資料p.32 関数呼び出し
887     if(p->n.op== POSTFIX){
888       parm_num = 0;
889       // printf("\n\npostfix bigin\nemit_postfix_expression/parm_num:%d,%s",
890       char address[64];未定義の関数を呼び出すときには、
891       //するたびにを指示してもよいcallextern
892       emit_argument_expression_list_opt(p->tp.a[1]);
893       parm_num=0;
894
895
896
897       // printf("\n\nemit_postfix_expression/parm_num:%d",parm_num);
898       if(p->tp.a[0]->tk.kind == UNDEFFUN)
899         emit("EXTERN", p->tp.a[0]->tk.name, NULL, NULL);引数
      がある場合、引数の中身を計算して
900       //するpush
901       if (p->tp.a[1]){//がでない場合argument_exp_listnull
902         tree t = p->tp.a[1];
903         while(t->tp.op == CONS){
904
905
906
907           parm_num++;
908           t = t->tp.a[0];
909         }
910
911
912         parm_num++;
913         parm_num_esp=parm_num;
914
915       }else{
916         parm_num_esp引数ないとき=0;//
917       }関数呼び出し
918       //
919       // printf("\n\nemit_postfix_expression/parm_num:%d",parm_num);
920       emit("call", p->tp.a[0]->tk.name, NULL, NULL);
921       sprintf(address, "%d", parm_num_esp * 4);
922       emit("add", "esp", address, NULL);
```

```
923
924
925     }else{
926       emit_primary_expr(p);
927
928     }
929   }
930
931
932   void emit_argument_expression_list_opt(tree p){
933     if(p==NULL){}
934     else{   parm_num=0;
935       emit_argument_expression_list(p);
936
937       //    printf("\n\n\nemit_argument_exp_list/parm_num:%d",parm_num);
938     }
939
940   }
941   //22
942   void emit_unary_expr(tree p){
943     //   printf("22unary_expr");
944     if(p->n.op== UNARY){
945       emit_unary_expr(p->tp.a[0]);
946       emit("imul","eax","-1",NULL);
947
948
949     }else{
950       emit_postfix_expr(p);
951
952     }
953
954   }
955   //23
956   void emit_primary_expr(tree p){
957     // printf("23primary_expr");
958     if(p->tk.op==TOKEN){
959
960       //   printf("IDENTIFIER");
961       //のshowidentifier の局所変数の場合
962       char loc[256] = "[ebp", offset[64];
963       sprintf(offset, "%+d", p->tk.offsetが符号つきの数字を表示
    );//%+
964       strcat(loc, offset);
```

```
965       strcat(loc, "]");コード生成
966
967       //
968       emit("mov", "eax", loc, NULL);
969
970       //      emit_parm(p);
971       // printf("%s",p->tk.name);//make_token_node
972     }else if(p->c.op ==CONSTANT){
973
974
975       char value[64];
976       sprintf(value, "%d", p->c.v);
977       emit("mov", "eax", value, NULL);
978       /*  printf("make_constant_node");
979           printf("%d",p->c.v);//make_constant_node
980       */
981     }else{
982       //printf("(");
983       emit_expression(p);
984       //printf(")\n");
985
986     }
987
988
989 }
990 //24
991 void emit_argument_expression_list(tree p){
992   //  printf("24argument_expression_list");
993   if(p->n.op!= CONS){
994     emit_assign_expr(p);
995     emit("push", "eax", NULL, NULL);
996     parm_num++;
997     parm_num_esp=parm_num++;
998     // printf("\n\nemit_argument_expression/parm_num:%d",parm_num);
999   }else{
1000    emit_assign_expr(p->tp.a[1]);
1001    emit("push", "eax", NULL, NULL);
1002    parm_num++;
1003    emit_argument_expression_list(p->tp.a[0]);
1004
1005    //  printf("\n\nemit_argument_expression/parm_num:%d",parm_num);
1006
1007  }
```

```
1008  }
```

2　感想