컴파일러 과제-7

20192800 권대현

1. 과제 내용

* 이번 컴파일러 과제는 과제-6에서 구현한 C언어 시멘틱 분석기에 더불어 9장에서 설명한 코드 생성기 프로그램을 완성하여 실험하는 것이다.
* 입력으로 다양한 선언문과 명령문을 포함하는 프로그램들이 주어진다.
* 수식이 잘못된 경우 line 번호와 함께 syntax 오류가 어디서 일어났는지를 출력한다.
* 수식이 올바를 경우 syntax tree를 출력하여 신택스 분석이 된 과정을 보여준다. 그 다음으로 semantic tree를 출력하여 시멘틱 분석이 된 과정을 보여준다.
* 수식이 잘못된 경우 에러 번호를 출력하여 어떤 에러가 발생했는지 알려준다.
* 수식이 올바를 경우 어셈블리어로 주어진 코드를 번역하고 a.asm 파일을 생성하고 코드를 저장한다.
* 생성된 a.asm 파일을 주어진 어셈블러 인터프리터인 interp.exe에 넘겨준다.
* 수식이 잘못된 경우 line 번호와 함께 syntax 오류가 어디서 일어났는지를 출력한다.
* 수식이 올바를 경우 추출된 symbol과 번호를 출력하고, 코드를 번호 순으로 출력하여 최종적으로 코드를 계산한 결과값을 출력한다.

1. 문제 및 해결 방법

* gen.c 구현 문제
  + 이번 과제에서 추가로 구현해야 할 사항은 분석이 끝난 소스 코드를 어셈블리어로 번역하는 함수들을 만드는 것이다. 이에 따라 컴파일러 pdf 9장, 10-11장과 9장 화상 강의를 참고하여 gen.c를 생성하였고, 내부 함수를 구현하는 도중에 몇몇 문제가 발생했다.
  + gen\_literal\_table() 구현 문제
    - 처음 해당 함수를 만들 때 i=1부터 semantic.c에 선언된 literal\_no까지 for문으로 반복하여 .literal 주소와 상수를 저장하도록 코드를 작성했다. 그러나, 상수를 저장할 때 쓰이는 value값이 union 타입인 줄 모르고, value값만 사용했더니 fprintf로 파일에 저장하는 함수에 오류가 생겼다.
    - 이에 literal\_table[] 배열을 참조하여 해당 멤버의 type값에 따라 value.i, value.f 등 다르게 저장할 수 있도록 타입에 따른 파일 저장을 구현하여 해결했다.
  + Typedef enum op OPCODE 사용 문제
    - 새로 받은 type.h에 어셈블리어 명령어를 포함하는 enum형 OPCODE가 선언되어 있다. 이를 참조하기 위해 extern enum op OPCODE나 typedef enum op OPCODE 등 여러 시도를 했지만 제대로 참조되지 못하는 문제가 발생했다.
    - 답은 간단했다. 새로 interp/type.h로 경로를 재설정하여 헤더파일을 참조하기만 하면 됐다. 오히려 따로 extern 키워드를 통해 불러올 필요도 없었다.
  + gen\_declaration\_list() 구현 문제
    - pdf 상으로 gen\_declaration() 함수는 예시 코드가 있지만, gen\_declaration\_list()는 존재하지 않아서 구현에 애를 먹게 됐다.
    - gen\_declaration\_list라는 함수 이름에서 알 수 있듯이, gen\_declaration()을 list로 받아서 한꺼번에 실행해주는 함수라는 것을 알고, while문으로 인자로 넘겨 받은 id를 넣어서 계속해서 다음 link로 gen\_declaration()을 수행할 수 있게끔 코드를 작성하여 해결했다.

1. 테스트

* 올바른 C언어 코드
  + Hello World 출력

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - Main 함수에서 printf(“Hello World!\n”)를 수행했다.
    - a.exe를 통해 Hello World를 출력하는 C코드를 어셈블리어로 번역하고 이를 interp.exe에 넣음으로써 어셈블러/인터프리터를 실행한다.
  + int \*fun() 함수

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* + - 파라미터로 int a를 넘겨받고 지역 변수로 int x를 선언하는 int \*fun()함수를 선언한다.
  + Enum 선언 테스트

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* + - Enum 형으로 color을 정의하고, color 내부 멤버를 초기화하는 코드를 선언했다.
    - Enum color c1, c2; 를 통해 사전 정의된 color 형을 사용하는 변수 선언을 테스트했다.
    - 마지막으로 printf로 enum red 값이 정수로 얼마인지를 출력했다.
  + Printf, scanf 테스트

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - 간단하게 scanf를 통해 사용자 입력을 받고, 이를 변수 i에 저장했다가 출력하는 코드를 테스트해봤다.
    - 정상적으로 1을 입력받고 1을 다시 출력했다.
  + 함수 선언 및 곱셈 테스트

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - 인자로 a와 b를 입력 받아서 이를 곱한 뒤 값을 리턴하는 함수 mult()를 선언했다.
    - Main 함수에서 int i를 선언하고 mult를 호출하여 a와 b 각각 5, 6을 넣어서 성공적으로 30이라는 값을 리턴 받고 이를 출력했다.
  + 더 복잡한 함수 및 int형 배열 테스트

전자제품, 텍스트, 디스플레이, 스크린샷이(가) 표시된 사진

자동 생성된 설명

* + - int형으로 크기가 10인 배열을 선언하여 무작위로 숫자를 부여하고, 이를 sort함수에서 값을 오름차순으로 정렬하여 출력하는 코드를 생성했다.
    - 초기값을 for문을 통해 출력하고 sort를 통해 정렬한 뒤 성공적으로 정렬된 값을 배열을 참조하여 출력하고 있다.
* 잘못된 C언어 코드
  + Struct 테스트

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + - Struct 타입으로 int i와 char c를 갖는 변수 s를 선언했다. 이후 main 함수에서 s.i = 10을 통해 구조체의 값을 설정하고 printf로 이를 출력하는 테스트 코드를 작성했다.
    - Syntax와 semantic 분석기에서는 struct에 대해 처리를 해줬기 때문에 무사히 통과했으나, 어셈블리어 번역기에서 struct에 대해 코드를 구현하지 않았기 때문에 error가 발생했다.

1. 원시프로그램

* gen.c
* #include <stdio.h>
* #include <string.h>
* #include "type.h"
* #include "interp/type.h"
* char \*opcode\_name[] = {"OP\_NULL", "LOD", "LDX", "LDXB", "LDA", "LITI",
* "STO", "STOB", "STX", "STXB", "SUBI", "SUBF", "DIVI", "DIVF", "ADDI", "ADDF",
* "OFFSET", "MULI", "MULF", "MOD", "LSSI", "LSSF", "GTRI", "GTRF", "LEQI", "LEQF",
* "GEQI", "GEQF", "NEQI", "NEQF", "EQLI", "EQLF", "NOT", "OR", "AND",
* "CVTI", "CVTF", "JPC", "JPCR", "JMP", "JPT", "JPTR",
* "INT", "INCI", "INCF", "DECI", "DECF", "SUP", "CAL", "ADDR",
* "RET", "MINUSI", "MINUSF", "CHK", "LDI", "LDIB", "SWITCH", "SWVALUE",
* "SWDEFAULT", "SWLABEL", "SWEND", "POP", "POPB"};
* typedef enum
* {
* SW\_VALUE,
* SW\_DEFAULT
* } SW\_KIND;
* typedef struct sw
* {
* SW\_KIND kind;
* int val;
* int label;
* } A\_SWITCH;
* void code\_generation(A\_NODE \*);
* void gen\_literal\_table();
* void gen\_program(A\_NODE \*);
* void gen\_expression(A\_NODE \*);
* void gen\_expression\_left(A\_NODE \*);
* void gen\_arg\_expression(A\_NODE \*);
* void gen\_statement(A\_NODE \*, int, int, A\_SWITCH[], int \*);
* void gen\_statement\_list(A\_NODE \*, int, int, A\_SWITCH[], int \*);
* void gen\_initializer\_global(A\_NODE \*, A\_TYPE \*, int);
* void gen\_initializer\_local(A\_NODE \*, A\_TYPE \*, int);
* void gen\_declaration\_list(A\_ID \*);
* void gen\_declaration(A\_ID \*);
* void gen\_code\_i(OPCODE, int, int);
* void gen\_code\_f(OPCODE, int, float);
* void gen\_code\_s(OPCODE, int, char \*);
* void gen\_code\_l(OPCODE, int, int);
* void gen\_label\_number(int);
* void gen\_label\_name(char \*);
* void gen\_error();
* int get\_label();
* int label\_no = 0;
* int gen\_err = 0;
* extern FILE \*fout;
* extern A\_TYPE \*int\_type, \*float\_type, \*char\_type, \*void\_type, \*string\_type;
* extern A\_LITERAL literal\_table[];
* extern int literal\_no;
* void code\_generation(A\_NODE \*node)
* {
* gen\_program(node);
* gen\_literal\_table();
* }
* void gen\_literal\_table()
* {
* int i;
* for (i = 1; i <= literal\_no; i++)
* {
* fprintf(fout, ".literal %5d  ", literal\_table[i].addr);
* if (literal\_table[i].type == int\_type)
* fprintf(fout, "%d\n", literal\_table[i].value.i);
* else if (literal\_table[i].type == float\_type)
* fprintf(fout, "%f\n", literal\_table[i].value.f);
* else if (literal\_table[i].type == char\_type)
* fprintf(fout, "%d\n", literal\_table[i].value.c);
* else if (literal\_table[i].type == string\_type)
* fprintf(fout, "%s\n", literal\_table[i].value.s);
* }
* }
* void gen\_program(A\_NODE \*node)
* {
* switch (node->name)
* {
* case N\_PROGRAM:
* gen\_code\_i(INT, 0, node->value);
* gen\_code\_s(SUP, 0, "main");
* gen\_code\_i(RET, 0, 0);
* gen\_declaration\_list(node->clink);
* break;
* }
* }
* void gen\_expression(A\_NODE \*node)
* {
* A\_ID \*id;
* A\_TYPE \*t;
* int i, ll;
* switch (node->name)
* {
* case N\_EXP\_IDENT:
* id = node->clink;
* t = id->type;
* switch (id->kind)
* {
* case ID\_VAR:
* case ID\_PARM:
* switch (t->kind)
* {
* case T\_ENUM:
* case T\_POINTER:
* gen\_code\_i(LOD, id->level, id->address);
* break;
* case T\_ARRAY:
* if (id->kind == ID\_VAR)
* gen\_code\_i(LDA, id->level, id->address);
* else
* gen\_code\_i(LOD, id->level, id->address);
* break;
* case T\_STRUCT:
* gen\_error(24, node->line, "T\_STRUCT");
* case T\_UNION:
* gen\_code\_i(LDA, id->level, id->address);
* i = id->type->size;
* gen\_code\_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
* break;
* default:
* gen\_error(11, id->line);
* break;
* }
* break;
* case ID\_ENUM\_LITERAL:
* gen\_code\_i(LITI, 0, id->init);
* break;
* default:
* gen\_error(11, node->line);
* break;
* }
* break;
* case N\_EXP\_INT\_CONST:
* gen\_code\_i(LITI, 0, node->clink);
* break;
* case N\_EXP\_FLOAT\_CONST:
* i = node->clink;
* gen\_code\_i(LOD, 0, literal\_table[i].addr);
* break;
* case N\_EXP\_CHAR\_CONST:
* gen\_code\_i(LITI, 0, node->clink);
* break;
* case N\_EXP\_STRING\_LITERAL:
* i = node->clink;
* gen\_code\_i(LDA, 0, literal\_table[i].addr);
* break;
* case N\_EXP\_ARRAY:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* // gen\_code\_i(CHK,0,node->llink->type->expr);
* if (node->type->size > 1)
* {
* gen\_code\_i(LITI, 0, node->type->size);
* gen\_code\_i(MULI, 0, 0);
* }
* gen\_code\_i(OFFSET, 0, 0);
* if (!isArrayType(node->type))
* {
* i = node->type->size;
* if (i == 1)
* gen\_code\_i(LDIB, 0, 0);
* else
* gen\_code\_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
* }
* break;
* case N\_EXP\_FUNCTION\_CALL:
* t = node->llink->type;
* i = t->element\_type->element\_type->size;
* if (i % 4)
* i = i / 4 \* 4 + 4;
* if (node->rlink)
* {
* gen\_code\_i(INT, 0, 12 + i);
* gen\_arg\_expression(node->rlink);
* gen\_code\_i(POP, 0, node->rlink->value / 4 + 3);
* }
* else
* gen\_code\_i(INT, 0, i);
* gen\_expression(node->llink);
* gen\_code\_i(CAL, 0, 0);
* break;
* case N\_EXP\_STRUCT:
* gen\_error(24, node->line, "N\_EXP\_STRUCT");
* case N\_EXP\_ARROW:
* gen\_expression(node->llink);
* id = node->rlink;
* if (id->address > 0)
* {
* gen\_code\_i(LITI, 0, id->address);
* gen\_code\_i(OFFSET, 0, 0);
* }
* if (!isArrayType(node->type))
* {
* i = node->type->size;
* if (i == 1)
* gen\_code\_i(LDIB, 0, 0);
* else
* gen\_code\_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
* }
* break;
* case N\_EXP\_POST\_INC:
* gen\_expression(node->clink);
* gen\_expression\_left(node->clink);
* t = node->type;
* if (node->type->size == 1)
* gen\_code\_i(LDXB, 0, 0);
* else
* gen\_code\_i(LDX, 0, 1);
* if (isPointerOrArrayType(node->type))
* {
* gen\_code\_i(LITI, 0, node->type->element\_type->size);
* gen\_code\_i(ADDI, 0, 0);
* }
* else if (isFloatType(node->type))
* gen\_code\_i(INCF, 0, 0);
* else
* gen\_code\_i(INCI, 0, 0);
* if (node->type->size == 1)
* gen\_code\_i(STOB, 0, 0);
* else
* gen\_code\_i(STO, 0, 1);
* break;
* case N\_EXP\_POST\_DEC:
* gen\_expression(node->clink);
* gen\_expression\_left(node->clink);
* t = node->type;
* if (node->type->size == 1)
* gen\_code\_i(LDXB, 0, 0);
* else
* gen\_code\_i(LDX, 0, 1);
* if (isPointerOrArrayType(node->type))
* {
* gen\_code\_i(LITI, 0, node->type->element\_type->size);
* gen\_code\_i(SUBI, 0, 0);
* }
* else if (isFloatType(node->type))
* gen\_code\_i(DECF, 0, 0);
* else
* gen\_code\_i(DECI, 0, 0);
* if (node->type->size == 1)
* gen\_code\_i(STOB, 0, 0);
* else
* gen\_code\_i(STO, 0, 1);
* break;
* case N\_EXP\_PRE\_INC:
* gen\_expression\_left(node->clink);
* t = node->type;
* if (node->type->size == 1)
* gen\_code\_i(LDXB, 0, 0);
* else
* gen\_code\_i(LDX, 0, 1);
* if (isPointerOrArrayType(node->type))
* {
* gen\_code\_i(LITI, 0, node->type->element\_type->size);
* gen\_code\_i(ADDI, 0, 0);
* }
* else if (isFloatType(node->type))
* gen\_code\_i(INCF, 0, 0);
* else
* gen\_code\_i(INCI, 0, 0);
* if (node->type->size == 1)
* gen\_code\_i(STXB, 0, 0);
* else
* gen\_code\_i(STX, 0, 1);
* break;
* case N\_EXP\_PRE\_DEC:
* gen\_expression\_left(node->clink);
* t = node->type;
* if (node->type->size == 1)
* gen\_code\_i(LDXB, 0, 0);
* else
* gen\_code\_i(LDX, 0, 1);
* if (isPointerOrArrayType(node->type))
* {
* gen\_code\_i(LITI, 0, node->type->element\_type->size);
* gen\_code\_i(SUBI, 0, 0);
* }
* else if (isFloatType(node->type))
* gen\_code\_i(DECF, 0, 0);
* else
* gen\_code\_i(DECI, 0, 0);
* if (node->type->size == 1)
* gen\_code\_i(STXB, 0, 0);
* else
* gen\_code\_i(STX, 0, 1);
* break;
* case N\_EXP\_NOT:
* gen\_expression(node->clink);
* gen\_code\_i(NOT, 0, 0);
* break;
* case N\_EXP\_PLUS:
* gen\_expression(node->clink);
* break;
* case N\_EXP\_MINUS:
* gen\_expression(node->clink);
* if (isFloatType(node->type))
* gen\_code\_i(MINUSF, 0, 0);
* else
* gen\_code\_i(MINUSI, 0, 0);
* break;
* case N\_EXP\_AMP:
* gen\_expression\_left(node->clink);
* break;
* case N\_EXP\_STAR:
* gen\_expression(node->clink);
* i = node->type->size;
* if (i == 1)
* gen\_code\_i(LDIB, 0, 0);
* else
* gen\_code\_i(LDI, 0, i % 4 ? i / 4 + 1 : i / 4);
* break;
* case N\_EXP\_SIZE\_EXP:
* gen\_code\_i(LITI, 0, node->clink);
* break;
* case N\_EXP\_SIZE\_TYPE:
* gen\_code\_i(LITI, 0, node->clink);
* break;
* case N\_EXP\_CAST:
* gen\_expression(node->rlink);
* if (node->type != node->rlink->type)
* if (isFloatType(node->type))
* gen\_code\_i(CVTF, 0, 0);
* else if (isFloatType(node->rlink->type))
* gen\_code\_i(CVTI, 0, 0);
* break;
* case N\_EXP\_MUL:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->type))
* gen\_code\_i(MULF, 0, 0);
* else
* gen\_code\_i(MULI, 0, 0);
* break;
* case N\_EXP\_DIV:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->type))
* gen\_code\_i(DIVF, 0, 0);
* else
* gen\_code\_i(DIVI, 0, 0);
* break;
* case N\_EXP\_MOD:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* gen\_code\_i(MOD, 0, 0);
* break;
* case N\_EXP\_ADD:
* gen\_expression(node->llink);
* if (isPointerOrArrayType(node->rlink->type))
* {
* gen\_code\_i(LITI, 0, node->rlink->type->element\_type->size);
* gen\_code\_i(MULI, 0, 0);
* }
* gen\_expression(node->rlink);
* if (isPointerOrArrayType(node->llink->type))
* {
* gen\_code\_i(LITI, 0, node->llink->type->element\_type->size);
* gen\_code\_i(MULI, 0, 0);
* }
* if (isFloatType(node->type))
* gen\_code\_i(ADDF, 0, 0);
* else
* gen\_code\_i(ADDI, 0, 0);
* break;
* case N\_EXP\_SUB:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isPointerOrArrayType(node->llink->type) && !isPointerOrArrayType(node->rlink->type))
* {
* gen\_code\_i(LITI, 0, node->llink->type->element\_type->size);
* gen\_code\_i(MULI, 0, 0);
* }
* if (isFloatType(node->type))
* gen\_code\_i(SUBF, 0, 0);
* else
* gen\_code\_i(SUBI, 0, 0);
* break;
* case N\_EXP\_LSS:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->llink->type))
* gen\_code\_i(LSSF, 0, 0);
* else
* gen\_code\_i(LSSI, 0, 0);
* break;
* case N\_EXP\_GTR:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->llink->type))
* gen\_code\_i(GTRF, 0, 0);
* else
* gen\_code\_i(GTRI, 0, 0);
* break;
* case N\_EXP\_LEQ:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->llink->type))
* gen\_code\_i(LEQF, 0, 0);
* else
* gen\_code\_i(LEQI, 0, 0);
* break;
* case N\_EXP\_GEQ:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->llink->type))
* gen\_code\_i(GEQF, 0, 0);
* else
* gen\_code\_i(GEQI, 0, 0);
* break;
* case N\_EXP\_NEQ:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->llink->type))
* gen\_code\_i(NEQF, 0, 0);
* else
* gen\_code\_i(NEQI, 0, 0);
* break;
* case N\_EXP\_EQL:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* if (isFloatType(node->llink->type))
* gen\_code\_i(EQLF, 0, 0);
* else
* gen\_code\_i(EQLI, 0, 0);
* break;
* case N\_EXP\_AND:
* gen\_expression(node->llink);
* gen\_code\_l(JPCR, 0, i = get\_label());
* gen\_expression(node->rlink);
* gen\_label\_number(i);
* break;
* case N\_EXP\_OR:
* gen\_expression(node->llink);
* gen\_code\_l(JPTR, 0, i = get\_label());
* gen\_expression(node->rlink);
* gen\_label\_number(i);
* break;
* case N\_EXP\_ASSIGN:
* gen\_expression\_left(node->llink);
* gen\_expression(node->rlink);
* i = node->type->size;
* if (i == 1)
* gen\_code\_i(STXB, 0, 0);
* else
* gen\_code\_i(STX, 0, i % 4 ? i / 4 + 1 : i / 4);
* break;
* default:
* gen\_error(100, node->line);
* break;
* }
* }
* void gen\_expression\_left(A\_NODE \*node)
* {
* A\_ID \*id;
* A\_TYPE \*t;
* int result;
* switch (node->name)
* {
* case N\_EXP\_IDENT:
* id = node->clink;
* t = id->type;
* switch (id->kind)
* {
* case ID\_VAR:
* case ID\_PARM:
* switch (t->kind)
* {
* case T\_ENUM:
* case T\_POINTER:
* case T\_STRUCT:
* case T\_UNION:
* gen\_code\_i(LDA, id->level, id->address);
* break;
* case T\_ARRAY:
* if (id->kind == ID\_VAR)
* gen\_code\_i(LDA, id->level, id->address);
* else
* gen\_code\_i(LOD, id->level, id->address);
* break;
* default:
* gen\_error(13, node->line, id->name);
* break;
* }
* break;
* case ID\_FUNC:
* gen\_code\_s(ADDR, 0, id->name);
* break;
* default:
* gen\_error(13, node->line, id->name);
* break;
* }
* break;
* case N\_EXP\_ARRAY:
* gen\_expression(node->llink);
* gen\_expression(node->rlink);
* // gen\_code\_i(CHK,0,node->llink->type->expr);
* if (node->type->size > 1)
* {
* gen\_code\_i(LITI, 0, node->type->size);
* gen\_code\_i(MULI, 0, 0);
* }
* gen\_code\_i(OFFSET, 0, 0);
* break;
* case N\_EXP\_STRUCT:
* gen\_expression\_left(node->llink);
* id = node->rlink;
* if (id->address > 0)
* {
* gen\_code\_i(LITI, 0, id->address);
* gen\_code\_i(OFFSET, 0, 0);
* }
* break;
* case N\_EXP\_ARROW:
* gen\_expression(node->llink);
* id = node->rlink;
* if (id->address > 0)
* {
* gen\_code\_i(LITI, 0, id->address);
* gen\_code\_i(OFFSET, 0, 0);
* }
* break;
* case N\_EXP\_STAR:
* gen\_expression(node->clink);
* break;
* case N\_EXP\_INT\_CONST:
* case N\_EXP\_FLOAT\_CONST:
* case N\_EXP\_CHAR\_CONST:
* case N\_EXP\_STRING\_LITERAL:
* case N\_EXP\_FUNCTION\_CALL:
* case N\_EXP\_POST\_INC:
* case N\_EXP\_POST\_DEC:
* case N\_EXP\_PRE\_INC:
* case N\_EXP\_PRE\_DEC:
* case N\_EXP\_NOT:
* case N\_EXP\_MINUS:
* case N\_EXP\_SIZE\_EXP:
* case N\_EXP\_SIZE\_TYPE:
* case N\_EXP\_CAST:
* case N\_EXP\_MUL:
* case N\_EXP\_DIV:
* case N\_EXP\_MOD:
* case N\_EXP\_ADD:
* case N\_EXP\_SUB:
* case N\_EXP\_LSS:
* case N\_EXP\_GTR:
* case N\_EXP\_LEQ:
* case N\_EXP\_GEQ:
* case N\_EXP\_NEQ:
* case N\_EXP\_EQL:
* case N\_EXP\_AMP:
* case N\_EXP\_AND:
* case N\_EXP\_OR:
* case N\_EXP\_ASSIGN:
* gen\_error(12, node->line);
* break;
* default:
* gen\_error(100, node->line);
* break;
* }
* }
* void gen\_arg\_expression(A\_NODE \*node)
* {
* A\_NODE \*n;
* switch (node->name)
* {
* case N\_ARG\_LIST:
* gen\_expression(node->llink);
* gen\_arg\_expression(node->rlink);
* break;
* case N\_ARG\_LIST\_NIL:
* break;
* default:
* gen\_error(100, node->line);
* break;
* }
* }
* int get\_label()
* {
* label\_no++;
* return (label\_no);
* }
* void gen\_statement(A\_NODE \*node, int cont\_label, int break\_label, A\_SWITCH sw[], int \*sn)
* {
* A\_NODE \*n;
* int i, l1, l2, l3;
* switch (node->name)
* {
* case N\_STMT\_LABEL\_CASE:
* case N\_STMT\_LABEL\_DEFAULT:
* break;
* case N\_STMT\_COMPOUND:
* if (node->llink)
* gen\_declaration\_list(node->llink);
* gen\_statement\_list(node->rlink, cont\_label, break\_label, sw, sn);
* break;
* case N\_STMT\_EMPTY:
* break;
* case N\_STMT\_EXPRESSION:
* n = node->clink;
* gen\_expression(n);
* i = n->type->size;
* if (i)
* gen\_code\_i(POP, 0, i % 4 ? i / 4 + 1 : i / 4);
* break;
* case N\_STMT\_IF:
* gen\_expression(node->llink);
* gen\_code\_l(JPC, 0, l1 = get\_label());
* gen\_statement(node->rlink, cont\_label, break\_label, 0, 0);
* gen\_label\_number(l1);
* break;
* case N\_STMT\_IF\_ELSE:
* gen\_expression(node->llink);
* gen\_code\_l(JPC, 0, l1 = get\_label());
* gen\_statement(node->clink, cont\_label, break\_label, 0, 0);
* gen\_code\_l(JMP, 0, l2 = get\_label());
* gen\_label\_number(l1);
* gen\_statement(node->rlink, cont\_label, break\_label, 0, 0);
* gen\_label\_number(l2);
* break;
* case N\_STMT\_SWITCH:
* break;
* case N\_STMT\_WHILE:
* l3 = get\_label();
* gen\_label\_number(l1 = get\_label());
* gen\_expression(node->llink);
* gen\_code\_l(JPC, 0, l2 = get\_label());
* gen\_statement(node->rlink, l3, l2, 0, 0);
* gen\_label\_number(l3);
* gen\_code\_l(JMP, 0, l1);
* gen\_label\_number(l2);
* break;
* case N\_STMT\_DO:
* l3 = get\_label();
* l2 = get\_label();
* gen\_label\_number(l1 = get\_label());
* gen\_statement(node->llink, l2, l3, 0, 0);
* gen\_label\_number(l2);
* gen\_expression(node->rlink);
* gen\_code\_l(JPT, 0, l1);
* gen\_label\_number(l3);
* break;
* case N\_STMT\_FOR:
* n = node->llink;
* l3 = get\_label();
* if (n->llink)
* {
* gen\_expression(n->llink);
* i = n->llink->type->size;
* if (i)
* gen\_code\_i(POP, 0, i % 4 ? i / 4 + 1 : i / 4);
* }
* gen\_label\_number(l1 = get\_label());
* l2 = get\_label();
* if (n->clink)
* {
* gen\_expression(n->clink);
* gen\_code\_l(JPC, 0, l2);
* }
* gen\_statement(node->rlink, l3, l2, 0, 0);
* gen\_label\_number(l3);
* if (n->rlink)
* {
* gen\_expression(n->rlink);
* i = n->rlink->type->size;
* if (i)
* gen\_code\_i(POP, 0, i % 4 ? i / 4 + 1 : i / 4);
* }
* gen\_code\_l(JMP, 0, l1);
* gen\_label\_number(l2);
* break;
* case N\_STMT\_CONTINUE:
* if (cont\_label)
* gen\_code\_l(JMP, 0, cont\_label);
* else
* gen\_error(22, node->line);
* break;
* case N\_STMT\_BREAK:
* if (break\_label)
* gen\_code\_l(JMP, 0, break\_label);
* else
* gen\_error(23, node->line);
* break;
* case N\_STMT\_RETURN:
* n = node->clink;
* if (n)
* {
* i = n->type->size;
* if (i % 4)
* i = i / 4 \* 4 + 4;
* gen\_code\_i(LDA, 1, -i);
* gen\_expression(n);
* gen\_code\_i(STO, 0, i / 4);
* }
* gen\_code\_i(RET, 0, 0);
* break;
* default:
* gen\_error(100, node->line);
* break;
* }
* }
* void gen\_statement\_list(A\_NODE \*node, int cont\_label, int break\_label, A\_SWITCH sw[], int \*sn)
* {
* switch (node->name)
* {
* case N\_STMT\_LIST:
* gen\_statement(node->llink, cont\_label, break\_label, sw, sn);
* gen\_statement\_list(node->rlink, cont\_label, break\_label, sw, sn);
* break;
* case N\_STMT\_LIST\_NIL:
* break;
* default:
* gen\_error(100, node->line);
* break;
* }
* }
* void gen\_initializer\_global(A\_NODE \*node, A\_TYPE \*t, int addr)
* {
* //구현 X
* }
* void gen\_initializer\_local(A\_NODE \*node, A\_TYPE \*t, int addr)
* {
* //구현 X
* }
* void gen\_declaration\_list(A\_ID \*id)
* {
* while (id)
* {
* gen\_declaration(id);
* id = id->link;
* }
* }
* void gen\_declaration(A\_ID \*id)
* {
* int i;
* A\_NODE \*node;
* switch (id->kind)
* {
* case ID\_VAR:
* if (id->init)
* if (id->level == 0)
* gen\_initializer\_global(id->init, id->type, id->address);
* else
* gen\_initializer\_local(id->init, id->type, id->address);
* break;
* case ID\_FUNC:
* if (id->type->expr)
* {
* gen\_label\_name(id->name);
* gen\_code\_i(INT, 0, id->type->local\_var\_size);
* gen\_statement(id->type->expr, 0, 0, 0, 0);
* gen\_code\_i(RET, 0, 0);
* }
* break;
* case ID\_PARM:
* case ID\_TYPE:
* case ID\_ENUM:
* case ID\_STRUCT:
* case ID\_FIELD:
* case ID\_ENUM\_LITERAL:
* case ID\_NULL:
* break;
* default:
* gen\_error(100, id->line);
* break;
* }
* }
* void gen\_error(int i, int ll, char \*s)
* {
* gen\_err++;
* printf("\*\*\* error at line %d: ", ll);
* switch (i)
* {
* case 11:
* printf("illegal identifier in expression \n");
* break;
* case 12:
* printf("illegal l-value expression \n");
* break;
* case 13:
* printf("identifier %s not l-value expression \n", s);
* break;
* case 20:
* printf("illegal default label in switch statement \n");
* break;
* case 21:
* printf("illegal case label in switch statement \n");
* break;
* case 22:
* printf("no destination for continue statement \n");
* break;
* case 23:
* printf("no destination for break statement \n");
* break;
* case 24:
* printf("not implemented %s for code generation\n", s);
* break;
* case 100:
* printf("fatal compiler error during code generation\n");
* break;
* default:
* printf("unknown \n");
* break;
* }
* }
* void gen\_code\_i(OPCODE op, int l, int a)
* {
* fprintf(fout, "\t%9s   %d, %d\n", opcode\_name[op], l, a);
* }
* void gen\_code\_f(OPCODE op, int l, float a)
* {
* fprintf(fout, "\t%9s   %d, %f\n", opcode\_name[op], l, a);
* }
* void gen\_code\_s(OPCODE op, int l, char \*a)
* {
* fprintf(fout, "\t%9s   %d, %s\n", opcode\_name[op], l, a);
* }
* void gen\_code\_l(OPCODE op, int l, int a)
* {
* fprintf(fout, "\t%9s   %d, L%d\n", opcode\_name[op], l, a);
* }
* void gen\_label\_number(int i)
* {
* fprintf(fout, "L%d:\n", i);
* }
* void gen\_label\_name(char \*s)
* {
* fprintf(fout, "%s:\n", s);
* }