

Code Generation Example

Tsan-sheng Hsu

tshsu@iis.sinica.edu.tw

`http://www.iis.sinica.edu.tw/~tshsu`

Warnings

- This set of slides contain a “pseudo code” for a very simple compiler.
 - This is an example, not the standard solution.
 - Assume only integers.
 - Other solutions exist.
- This example does not pass LEX, YACC or GCC.
- Usage of this example is entirely to illustrate the high level ideas.
- Not responsible for any syntax errors.
- Please read LEX, YACC and GCC manuals carefully to avoid programming mistakes and conflicts.
 - web sites
 - ▷ *example: <http://dinosaur.compilertools.net/>*
 - libraries
 - ...

Error handling

```
char *error_message[MAXMSG]; /* tests for error messages */
void error_msg(int line_no; char * file_name;
int index; char *msg1; char *msg2)
{
    switch index {
        /* some messages needed to be specially treated */
        case 3: /* insert *msg1 into error message */
            /* variable 'name' undefined */
            printf("line %d in file %s, error %d, %s\n",
                line_no,file_name,index,error_message[index]);
            ...
        default:
            if(index >= MAXMSG){
                printf("internal error, wrong error index %d\n",index);
            }else{
                printf("line %d in file %s, error %d, %s\n",
                    line_no,file_name,index,error_message[index]);
            }
            exit(1);}
}
```

Global constants and variables

```
/* tags for where variables are stored */
#define GLOBAL_VAR 1
#define LOCAL_VAR 2
#define TEMP_VAR 3
#define REG_VAR 4
#define PARA_VAR 5
#define NON_LOCAL_VAR 6
#define CONSTANT_VAR 7
...

int global_start; /* the starting address to put global variables */
int local_start; /* the starting address to put local variables */
int temp_start; /* the starting address to put temp variables */
...
```

Type definitions

```
typedef struct place_typ {
    char tag; /* where this var is stored */
    int offset; /* store offset in an storage area for variables;
                store the constant value for constants */
    int depth_diff; /* difference in nesting depth */
} PLACE_TYPE;
```

```
typedef struct var_typ {
    char type_tag; /* procedure, variable, int constant ... */
    PLACE_TYPE place;
    char *name;
    /* for a procedure, store the following */
    int temp_start, local_start, param_start, ...;
    ...
} VAR_TYPE;
```

Code generation routines (1/2)

```
/* op: operator, opni: operand # i */
void emitASSIGN(char operator; PLACE_TYPE opn1,opn2,opn3)
{
    switch operator {
        case '+':
            gen_r_address(opn2,3); /* load into register I__3 */
            gen_r_address(opn3,4); /* load into register I__4 */
            fprintf(code_file,"I__3 =  I__3+ I__4;\n");
            gen_l_address(opn1,3);
            break;
        case '-': ...
        case '*': ...
        case '/': ...
        default:
            printf("internal error in code generation, operator
            %c is undefined\n",operator); exit(1);
    }
}
```

Code generation routines (2/2)

```
void emitCOND_JUMP(...)
{
  ...
}
void emitIO(...)
{
  ...
}
void emitCALL(...)
{
  ...
}
...
```

Generate *r*-address

```
/* load the value at 'where' into register # result_r */
void gen_r_address(PLACE_TYPE where; int result_r)
{
    switch where.tag {
    case GLOBAL_VAR:
        fprintf(code_file, "I__%1d = AVAL__S(%d);\n",
                result_r, global_start+where.offset);
        break;
    case LOCAL_VAR:
        /* make sure I__9 is not used for other purposes */
        fprintf(code_file, "I__9=I__%1d+%d\n", FP, local_start+where.offset);
        fprintf(code_file, "I__%1d = AVAL__S(I__9);\n", result_r,);
        break;
    case CONSTANT_VAR:
        fprintf(code_file, "I__%1d = %d;\n", result_r, where.offset); break;
        ...
    }
}
```


Generate *l*-address

```
/* store the value at register # result_r into the place ‘‘where’’ */
void gen_l_address(PLACE_TYPE where; int result_r)
{
    switch where.tag {
    case GLOBAL_VAR:
        fprintf(code_file, "ASSET__S(%d, I__%1d); \n",
                global_start+where.offset, result_r);
        break;
    case LOCAL_VAR:
        /* make sure I__9 is not used for other purposes */
        fprintf(code_file, "I__9=I__%1d+%d \n", FP, local_start+where.offset);
        fprintf(code_file, "ASSET__S(I__9, I__%1d); \n", result_r);
        break;
    case TEMP_VAR:
        ...
    default: /* internal error */
    }
}
```

TEMP space management (1/2)

```
#define MAXTEMP 1000
char temp_map[MAXTEMP];
static int max_temp_used; /* max number of temps used */
static int current_temp_used; /* current number of temps used */

void freeALLtemp(void)
{
    int i;

    max_temp_used = current_temp_used = 0;
    for(i=0;i<MAXTEMP;i++)
        temp_map[i] = 0;
}

void freetemp(PLACE_TYPE vplace)
{
    if(vplace.tag == TEMP_VAR){
        temp_map[vplace.offset] = 0;
        current_temp_used++;
    }
}
```

TEMP space management (2/2)

```
/* a very simple first fit allocation algorithm */
int newtemp(void)
{
    int i=0;

    while(i < MAXTEMP && temp_map[i] > 0) i++;
    if(i >= MAXTEMP){
        /* internal error, allocate more temps */
    }else{
        /* record the max number of temp space used in a procedure */
        current_temp_used++;
        if(current_temp_used > max_temp_used)
            max_temp_used = current_temp_used;
        temp_map[i] = 1;
        return(i);
    }
}
```

- Use the same bit map technique to record the set of registers used/unused.
- Check for free register first, then temp space.

Label management

```
static int current_label; /* current label number */

void initLABELS(void)
{
    current_label = 0;
}

int newLABEL(void)
{
    return(current_label++);
}

void emitLABEL(int label)
{
    fprintf(code_file, "LAB%d:\n", label);
}

void emitJUMP(int label)
{
    fprintf(code_file, "goto LAB%d;\n", label);
}
```

YACC rules

```
%union{
    int ival;
    double fval;
    VAR_TYPE vval;
}
/* define the return type of ‘‘expr’’ to be VAR_TYPE */
%type <vval> expr
...
expr : expr '+' expr
      {
          $$ .place.offset = newtemp();  $$ .place.tag = TEMP_VAR;
          emitASSIGN('+', $$ .place, $1 .place, $2 .place);
          freetemp($1 .place);          freetemp($2 .place);  ...
      }
    | expr '-' expr  {...}
    ...
    | id {...}
...

```