CS3300 Compiler Design (July 2024)

Assignment 2

Due October 8 23:59 on moodle

Objective:

This assignment aims to generate a Three-Address Code (TAC) from C-like language following specific rules. This will help understand intermediate code generation, a crucial step in the compilation process.

Supported Constructs:

In the constructs below, **"optional"** means that we will not have a test case to test it.

1. Types: int and strings, as variables, function parameters, and constants. Type char is supported only to enable strings (**e.g., str[0] = 'A';**), but otherwise, char need not be supported (**optional**). Integer arrays need not be supported (**optional**). Passing strings as function parameters should be supported (**needed**).
2. Declaration: global and local at the start of a function. No variables within any other scopes (**optional**). Global variables, if any, precede all the functions in the input file.
3. All identifiers (variables, functions) have the standard C format **[a-zA-Z_][a-zA-Z_0-9]***.
4. Expressions: integers with **+, -, *, /, **, ()**. For string[index] assignment, a constant char such as 'A' is a valid expression. Integer expressions need to be split as per precedence (same as in A1) for generating TAC. char and int types will not be combined (e.g., 'a' + 2).
5. Assignments: lvalue = expression; lvalue can be an integer variable or a string[index]. During function calls, an implicit assignment of string to a formal argument array should be supported. Support for passing string[index] as a function parameter is **optional**. Supporting assignment of integer to string[index] is **optional**.
6. Conditionals: if, if-else, while, for are supported.
7. The conditions will have **<, <=, >, >=, ==, !=** and support 0 or more logical **&&, ||, and !**.
8. The conditions will be only with integer expressions. No strings or characters will be used in conditions.

9. You need to support short-circuiting in this assignment.
10. Every function has the return type of int (no void). A return statement may or may not be present across all the exit paths of a function. A return statement may return an expression.
11. User-defined functions are supported. All user-defined functions should be defined before main. Their return type will be int. They may have zero or more arguments of type int or char array (string). An argument could be an expression. Function names will not be of the form L# or t#.
12. Comments are supported. They start with // and cover the remaining line. Supporting /* ... */ is optional.
13. The input programs will be syntactically valid as per this assignment's syntax. You don't need to check for syntax errors now.
14. The programs will also be semantically valid except for the following – which needs to be checked by your compiler:

a. If a variable is used but not defined (in the local and the global scopes as per static scoping), then issue an error: `undefined variable <varname>`

15. A variable with the same name may be defined across functions, but the global variables will always be unique (that is, they will not be redefined inside functions).
16. You can support anything additional over what is expected out of this assignment. But do not perform optimizations which will change the expected output.

Rules for Three-Address Code (TAC) Generation:

1. Types: Update symbol table. No TAC generated.

```
int x;
char str[10];
```

2. Assignment

Format:

```
t# = value;
var = t#
```

Description: Value should be assigned to a temporary variable first. Then assigned to the variable.

Example:

```
t1 = 5
a = t1
```

3. Expression Assignment

Format:

```
t# = Expression
var = t#
```

Description: Evaluate an expression and store the result in temporary variables, then assign the appropriate temporary variable to the target variable.

Example:

```
z = a + b * c ** d / (2 + 3);
becomes:
t1 = c ** d
t2 = b * t1
t3 = 2 + 3
t4 = t2 / t3
t5 = a + t4
z = t5
```

4. If-else Structure

Format:

```
if (condition) goto L#      //GOTO TRUE PART
goto L#                     //GOTO ELSE PART/EXIT LABEL
L#                          //IF PART
     // code for true part
     goto L#              //GOTO EXIT LABEL
L#:                        //ELSE PART
     // code for false part
L#:                        //EXIT LABEL
```

Description:

- Use conditional jumps to manage the if-else structure.
- L# denotes a label followed by a number, where the number starts from 1 and increments across the whole program (thus, labels are unique in the program).
- A temporary variable can be used with an integer value(applicable for && and ||)
  - 0 for false
  - 1 for true

Example:

`if (a < b) {`
`  a = b;`
`} else {`
`  b = a;`
`}`
	`t1 = a < b`
	`if (t1) goto L1`
	`goto L2`
	`L1:  // label numbers may change`
	`t2 = b`
	`a = t2`
	`goto L3`
	`L2:`
	`t3 = a`
	`b = t3`
	`L3:`

`if (a < b && b < c) {`
`  result = 1;`
`} else {`
`  result = 0;`
`}`
	`t1 = a < b`
	`if (t1) goto L1  //GOTO NEXT COND.`
	`goto L2  //GOTO ELSE`
	`L1:`
	`t2 = b < c`
	`if (t2) goto L3  //GOTO IF`
	`goto L2  //GOTO ELSE`
	`L3:  //IF PART`
	`t3 = 1`
	`result = t3`
	`goto L4`
	`L2:  //ELSE PART`
	`t4 = 0`
	`result = t4`
	`L4:`

`if (a < b || b < c) {`
`  result = 1;`
`} else {`
`  result = 0;`
`}`
	`t1 = a < b`
	`if (t1) goto L1 //short-circuit`
	`t2 = b < c`
	`if (t2) goto L1`
	`goto L2`
	`L1:`
	`t3 = 1  // IF PART`
	`result = t3`
	`goto L3`
	`L2:  //ELSE PART`
	`t4 = 0`
	`result = t4`
	`L3:`

`if (!(a > b)) {`
`  result = 1;`
`} else {`
`  result = 0;`
`}`
	`t1 = a > b`
	`t2 = not t1`
	`if (t2) goto L1`
	`goto L2`
	`L1:`
	`t1 = 1`
	`result = t1`
	`goto L3`
	`L2:`
	`t2 = 0`
	`result = t2`
	`L3:`

5. While Loop

Format:

```
L#:                                   // TOP OF LOOP
if (condition) goto L#                // GOTO TO LOOP BODY
      goto L#                         // GOTO END OF LOOP
L#:                                   // LOOP BODY LABEL
      // code in the loop section
      goto L#                         // GO TO TOP OF LOOP
L#:                                   // END OF LOOP
```

Description: Use a label to mark the beginning of the loop and conditional jumps to control the loop.

Example:

`while (i < 10) {`
`  i = i + 1;`
`}`
	`L1:`
	`t1 = i < 10`
	`if (t1) goto L2`
	`goto L3`
	`L2:`
	`t2 = i + 1`
	`i = t2`
	`goto L1`
	`L3:`

6. For Loop

Syntax: for (0 or 1 assignment; condition; 0 or 1 assignment) body with or without braces

Format:

```
Initialization (similar to Rule 2: Assignment)
L#:
if (condition) goto L#
goto L#
L#:
// code for the loop section
Increment/Decrement
goto L#
L#:
```

Description: Break down the for loop into initialization, condition check, loop body, and increment/decrement.

Example:

`for (i = 0; i < 5; i = i + 1) {`
`  // loop body`
`}`
	`i = 0`
	`L1:`
	`t1 = i < 5`
	`if (t1) goto L2`
	`goto L3`
	`L2:`
	`// loop body`
	`t2 = i + 1`
	`i = t2`
	`goto L1`
	`L3:`

7. Functions

Format:

Function_Name: (acts as a label)
// code in the function
retval = value
return

Description: Functions are treated as labels, and the return statement always returns a value.

Example:

`int func() {`
`  return 0;`
`}`
	`func:`
	`retval = 0`
	`return`

`int func(int a,int b) {`
`  a = a+b;`
`  return a;`
`}`
	`func:`
	`a = param1`
	`b = param2`
	`t1 = a+b`
	`a = t1`
	`retval = a`
	`return`

8. Function Call

Format:

```
param# = value | Expression
call Function_Name
```

Description: Parameters are assigned and then the function is called.
Example:

`func(a, b);`	`t1 = a`
	`t2 = b`
	`param1 = t1`
	`param2 = t2`
	`call func`

`fun(a + 2, b + c);`	`t1 = a + 2`
	`t2 = b + c`
	`param1 = t1`
	`param2 = t2`
	`call fun`

`fun1(fun2(a+b),fun3(c+d));`	`t1 = a + b`
	`param1 = t1`
	`call fun2`
	`t2 = retval`
	`t3 = c + d`
	`param1 = t3`
	`call fun3`
	`t4 = retval`

	param1 = t2 param2 = t4 call fun1

9. String Handling

Format: `t# = "some string"`
Description: Strings are assigned to temporary variables, mainly for use in printf.
Example: `t1 = "Hello World!"`

10. Calling printf

Format:

```
param1 = t#
param2 = …
call printf
```

Description: printf is called with comma-separated parameters.
Example:

printf("Result: %d", x);	t1 = "Result: %d" t2 = x param1 = t1 param2 = t2 call printf

Notes:
Labels (L#, etc.) and temporary variables (t#) are used to maintain the flow and handle intermediate computations.
Make sure to number temporary variables and labels sequentially (from the first function to main) to avoid conflicts.

Examples:

int main() {
  int a;
  a = 5;
  return 0;
	main:
	t1 = 5
	a = t1
	retval = 0

} 	return

int main() {
    int a;
    int b;
    int c;
    a = 5;
    b = 10;
    c = a + b;
    return 0;
}
 	
	main:
	t1 = 5
	a = t1
	t2 = 10
	b = t2
	t3 = a + b
	c = t3
	retval = 0
	return

int main() {
    int a = 5;
    int b = 10;
    if (a < b) {
        a = b;
    } else {
        b = a;
    }
    return 0;
}
 	
	main:
	t1 = 5
	a = t1
	t2 = 10
	b = t2
	t3 = a < b
	if (t3) goto L1
	goto L2
	L1:
	t3 = b
	a = t3
	goto L3
	L2:
	t4 = a
	b = t4
	L3:
	retval = 0
	return

int result; // Global variable
int flag = 0;
 	
int add(int x, int y) {
    result = x + y;
    return x + y;
}
 	
int main() {
    int a = 5;
    int b = 10;
    int c;
 	
	global result
	global flag
	t1 = 0
	flag = t1
	
	add:
	x = param1
	y = param2
	t1 = x + y
	result = t1
	t2 = x + y
	retval = t2
	return

`    c = add(a, b);`
`    return 0;`
`}`
	`main:`
	`t3 = 5`
	`a = t3`
	`t4 = 10`
	`b = t4`
	`param1 = t3`
	`param2 = t4`
	`call add`
	`c = retval`
	`retval = 0`
	`return`

`int main() {`
`    int i;`
`    i = 0;`
`    while (i < 10) {`
`        i = i + 1;`
`    }`
`    return 0;`
`}`
	`main:`
	`t1 = 0`
	`i = t1`
	`L1:`
	`t2 = i < 10`
	`if (t2) goto L2`
	`goto L3`
	`L2:`
	`t3 = i + 1`
	`i = t3`
	`goto L1`
	`L3:`
	`retval = 0`
	`return`

`int main() {`
`    int i;`
`    for (i = 0; i < 5; i = i + 1) {`
`        printf("Iteration %d", i);`
 	
`    }`
`    return 0;`
`}`
	`t1 = 0`
	`i = t1`
	`L1:`
	`t2 = i < 5`
	`if (t2) goto L2`
	`goto L3`
	`L2:`
	`t3 = "Iteration %d"`
	`t4 = i`
	`param1 = t3`
	`param2 = t4`
	`call printf`
	`t5 = i + 1`
	`i = t5`
	`goto L1`
	`L3:`

	retval = 0 return

int main() {
  int data;
  data = 100;
 	
  int i;
  for (i = 0; i < 10; i = i + 1) {
  data = data + 1;
  printf("Bye Bye %d", i);
  }
 	
  if (data > 111) {
  if (i < 10) {
  printf("Failed");
  } else {
  printf("Success");
  }
  }
 	
  return 0;
}
	main
	t1 = 100:
	data = t1
	t2 = 0
	i = t2
	L1:
	t2 = i < 10
	if (t3) goto L2
	goto L3
	L2:
	t4 = data + 1
	data = t4
	t5 = "Bye Bye %d"
	t6 = i
	param1 = t5
	param2 = t6
	call printf
	t7 = i + 1
	i = t7
	goto L1
	L3:
	t8 = data > 111
	if (t8) goto L4
	goto L5
	L4:
	t9 = i < 10
	if (t9) goto L6
	goto L7
	L6:
	t10 = "Failed"
	param1 = t10
	call printf
	goto L8
	L7:
	t11 = "Success"
	param1 = t11
	call printf
	L8:
	retval = 0
	return

Submission Guidelines:

1. Create a new directory and rename it to your roll number, in the format CSXXBXXX (e.g., CS12B345).
2. Ensure that (a2.l and a2.y) files related to your project are placed inside this CSXXBXXX directory and there is no other directory inside CSXXBXXX.
3. Write a Makefile and place it inside the CSXXBXXX directory. The executable must be **a.out** (we will follow this for all the future labs + assignments, so there is no confusion).
4. Zip the directory using the following command:
   zip CSXXBXXX.zip -r CSXXBXXX
5. Unzipping CSXXBXXX.zip must give a folder CSXXBXXX containing only source files (a2.l, a2.y, your C/C++ files and Makefile). It would help if you check this after submitting on moodle.
6. For any queries about the assignment, please post on moodle, so everyone can know about it.