

COSC 320 Principles of Programming Languages

Bonus Documentation

Lina Benna
100063450

Farah Hussain Hassan
100061366

1 Do-While Implementation

We decided to include handling for a do-while loop in the Rust translation.

The primary difference rested in the control flow. The do-while loop executes the body at least once before evaluating the loop condition, whereas a while loop checks the condition before the first execution. In c4, the while loop is structured to check the condition first and then branch based on its result. In this implementation, the do-while loop must first execute the body and then check the condition in the while statement. This change is reflected in the jump instruction used: while the while loop uses BZ (branch if zero/false) to jump when the condition is false, the do-while loop uses BNZ (branch if not zero/true) because it needs to repeat the loop body as long as the condition remains true.

Moreover, we replaced pointers with safer, more structured types like Vec, which automatically handled memory allocation and resizing. For example, instead of manually incrementing the instruction pointer, we pushed new instructions to a Vec (e.g: `self.e.push(BNZ)`), and the vector dynamically grew as needed. The Rust compiler ensured that no out-of-bounds accesses occurred, and it optimized memory usage by reusing space and handling reallocations safely.

2 Source Code

```
1 Token::Do => {
2     self.next();
3     a = self.e.len() + 1; // Beginning of do-while body
4     self.stmt();
5     if let Token::While = self.tk {
6         self.next();
7         if let Token::LParen = self.tk {
8             self.next();
9         } else {
10            eprintln!("{}", "open paren expected after while in do-while",
11                           self.line);
12            std::process::exit(-1);
13        }
14        self.expr(Token::Assign.precedence().unwrap());
15        if let Token::RParen = self.tk {
16            self.next();
17        } else {
18            eprintln!("{}", "close paren expected after while in do-while",
19                           self.line);
20            std::process::exit(-1);
21        }
22    }
23 }
```

```

19     }
20     self.e.push(BNZ);
21     self.e.push(a as i32);
22     if let Token::Semicolon = self.tk {
23         self.next();
24     } else {
25         eprintln!("{}", semicolon expected after do-while", self.line)
26         ;
27         std::process::exit(-1);
28     } else {
29         eprintln!("{}", while expected after do statement", self.line);
30         std::process::exit(-1);
31     }
32 }

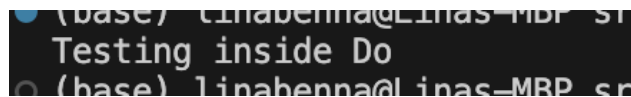
```

3 do-while.c

```

1 #include <stdio.h>
2
3 int main() {
4     do {
5         printf("Testing inside Do \n");
6     } while (0);
7     return 0;
8 }

```



A terminal window showing the execution of the do-while loop. The prompt is (base) linahenna@Linus-MBP. The command executed is ./do-while.c, and the output is Testing inside Do. The prompt is then (base) linahenna@Linus-MBP.

Figure 1: Do-While Implementation.