

# Compiler Milestone 3

Abir Rajbongshi - 210273, Aditya Bangar - 210069, Pratham Sahu - 210755

Course Instructor : Dr Swarnendu Biswas

**Abstract**—This is the report for the final milestone of CS335-Compiler Design course.

## 1. Compiling the Compiler

The directory structure is as follows:

```
/
├── Makefile
├── run.sh
├── run_x86.sh
├── src/
│   ├── allocmem.s
│   ├── lexer.l
│   ├── node.cpp
│   ├── parser.y
│   ├── print.s
│   ├── symbol_table.cpp
│   └── x86.cpp
└── testcases/
    ├── testcase1.py
    ├── testcase2.py
    ├── testcase3.py
    ├── testcase4.py
    └── testcase5.py
```

### commands to run

1. make
2. ./run.sh testcases/testcase1.py
3. gcc -o a.out x86\_code.s -no-pie
4. ./a.out

The above commands will generate the x86\_code.s file which contains the assembly code for the given python file along with the other files from the previous milestones. The a.out file will be the executable file for the given python file.

## 2. Language Features Implemented

### 2.1. Minimum Features to be implemented :

- **Arithmetic Operations** : Addition(+), Subtraction(-), Multiplication(\*), Division(/, //), Modulus(%)
- **Relational Operations** : Less than(<), Less than or equal to(<=), Greater than(>), Greater than or equal to(>=), Equal to(=), Not equal to(!=)
- **Logical Operations** : And(and), Or(or), Not(not)
- **(Bitwise Operations)** : And(&), Or(|), Xor(^), Left shift(«), Right shift(»)
- **Assignment Operations** : Simple assignment(=), Add and assign(+=), Subtract and assign(-=), Multiply and assign(\*=), Divide and assign(/=), Modulus and assign(%=), Bitwise and assign(&=), Bitwise or assign(|=), Bitwise xor assign(^=), Left shift and assign(«=), Right shift and assign(»=)
- **Control Structures** : if-else, if-elif-else, while, for, break, continue
- **Data Types** : Integers, Booleans, Strings, Arrays
- **Functions** : Function definition, Function call, Function return, Function arguments, Function recursion, library functions (len, print, range)

- **Classes** : Class definition, Class instantiation, Class methods (static and non-static), Class inheritance, Class attributes, Objects, Self
- **Multilevel Inheritance** : Implemented multilevel inheritance in the language.
- **String Comparison** : Implemented string comparison in the language.

### 2.2. Additional Features Implemented :

- **Functions inside Functions** : Cases like

```
print(func(a, b))
print(func1(a, b) + func2(func1(a, b) + c))
range(len(arr))
```

are handled.

- **Array Handling and Extensions** : Cases like

```
arr = [1, 2, 3, 4, 5]
# passing arrays as argument to function.
func(arr)
# returning arrays from functions.
arr2 = function_which_returns_array()
a : A = A()
b : B = B()
c : C = C()
arr3 = [a, b, c] # array of objects.
x : A = arr3[0]
x.func() # function call on object taken from
```

are also handled.

- **Passing Objects as Function Arguments** : Objects can be passed as arguments to functions.
- **List of objects**: We can have a list of non primitive data types(ex objects of a class)

## 3. Some Assumptions and Important Points

Some of the assumptions which we make while implementing the language are as follows:

- You cannot call len for an array that has been passed as a parameter. If you wish to use len, you can pass it as another parameter. len must be called only from the scope of the function where it is initialised
- We do not take care of arrays that have just one element. It does not make sense to support it, thus for ease of implementation of our compiler, we do not support it.

## 4. Examples

### Example 1

```
1 class Dog:
2     def __init__(self, name: str) -> None:
3         self.name: str = name
4
5     def speak(self) -> None:
6         print("woof")
7
8 class Pet(Dog):
9     def __init__(self, name: str, breed: str) -> None:
10        self.name: str = name
```

```

11 class Labrador(Pet):
12     def __init__(self, name: str, breed: str) -> None:
13         self.name: str = name
14         self.breed: str = breed
15
16
17 def main() -> None:
18     labrador: Labrador = Labrador("Max", "Labrador
19     Retriever")
20     labrador.speak()           # Output: Max says Woof!
21
22
23 if __name__ == "__main__":
24     main()

```

Code 1. Multilevel Inheritance

```

1 .data
2     integer_format: .asciz "%d\n"
3     string_format: .asciz "%s\n"
4     .str0:
5         .string "woof"
6     .str1:
7         .string "Max"
8     .str2:
9         .string "Labrador Retriever"
10    .str3:
11        .string "__main__"
12    .global main
13    .text
14 Dog.__init__:
15     pushq %rbp
16     movq %rsp, %rbp
17     pushq %rbx
18     pushq %r12
19     pushq %r13
20     pushq %r14
21     pushq %r15
22     subq $56, %rsp
23     movq 16(%rbp), %rdx
24     add $0, %rdx
25     movq %rdx, -40(%rbp)
26     movq -40(%rbp), %rax
27     movq 24(%rbp), %rdx
28     movq %rdx, (%rax)
29     addq $56, %rsp
30     popq %r15
31     popq %r14
32     popq %r13
33     popq %r12
34     popq %rbx
35     popq %rbp
36     ret
37 Dog.speak:
38     pushq %rbp
39     movq %rsp, %rbp
40     pushq %rbx
41     pushq %r12
42     pushq %r13
43     pushq %r14
44     pushq %r15
45     subq $64, %rsp
46     lea .str0(%rip), %rax
47     movq %rax, -48(%rbp)
48     movq $0, %rdi
49     pushq %rax
50     pushq %rdi
51     pushq %rsi
52     pushq %rdx
53     pushq %rcx
54     pushq %r8
55     pushq %r9
56     pushq %r10
57     pushq %r11
58     pushq -48(%rbp)
59     call global.print
60     addq $8, %rsp
61     popq %r11
62     popq %r10
63     popq %r9
64     popq %r8
65     popq %rcx
66     popq %rdx
67     popq %rsi
68     popq %rdi
69     popq %rax
70     movq $1, %rdi

```

```

71     addq $64, %rsp
72     popq %r15
73     popq %r14
74     popq %r13
75     popq %r12
76     popq %rbx
77     popq %rbp
78     ret
79 Pet.__init__:
80     pushq %rbp
81     movq %rsp, %rbp
82     pushq %rbx
83     pushq %r12
84     pushq %r13
85     pushq %r14
86     pushq %r15
87     subq $56, %rsp
88     movq 16(%rbp), %rdx
89     add $0, %rdx
90     movq %rdx, -40(%rbp)
91     movq -40(%rbp), %rax
92     movq 24(%rbp), %rdx
93     movq %rdx, (%rax)
94     addq $56, %rsp
95     popq %r15
96     popq %r14
97     popq %r13
98     popq %r12
99     popq %rbx
100    popq %rbp
101    ret
102 Labrador.__init__:
103    pushq %rbp
104    movq %rsp, %rbp
105    pushq %rbx
106    pushq %r12
107    pushq %r13
108    pushq %r14
109    pushq %r15
110    subq $72, %rsp
111    movq 16(%rbp), %rdx
112    add $0, %rdx
113    movq %rdx, -40(%rbp)
114    movq -40(%rbp), %rax
115    movq 24(%rbp), %rdx
116    movq %rdx, (%rax)
117    movq 16(%rbp), %rdx
118    add $8, %rdx
119    movq %rdx, -56(%rbp)
120    movq -56(%rbp), %rax
121    movq 32(%rbp), %rdx
122    movq %rdx, (%rax)
123    addq $72, %rsp
124    popq %r15
125    popq %r14
126    popq %r13
127    popq %r12
128    popq %rbx
129    popq %rbp
130    ret
131 main:
132    pushq %rbp
133    movq %rsp, %rbp
134    pushq %rbx
135    pushq %r12
136    pushq %r13
137    pushq %r14
138    pushq %r15
139    subq $120, %rsp
140    pushq %rax
141    pushq %rdi
142    pushq %rsi
143    pushq %rdx
144    pushq %rcx
145    pushq %r8
146    pushq %r9
147    pushq %r10
148    pushq %r11
149    pushq $32
150    call memalloc
151    add $8, %rsp
152    movq %rax, -48(%rbp)
153    popq %r11
154    popq %r10
155    popq %r9
156    popq %r8
157    popq %rcx
158    popq %rdx
159    popq %rsi
160    popq %rdi
161    popq %rax

```

```

162 movq $1, %rdi
163 lea .str2(%rip), %rax
164 movq %rax, -64(%rbp)
165 lea .str1(%rip), %rax
166 movq %rax, -80(%rbp)
167 pushq %rax
168 pushq %rdi
169 pushq %rsi
170 pushq %rdx
171 pushq %rcx
172 pushq %r8
173 pushq %r9
174 pushq %r10
175 pushq %r11
176 pushq -64(%rbp)
177 pushq -80(%rbp)
178 pushq -48(%rbp)
179 call Labrador.__init__
180 addq $24, %rsp
181 popq %r11
182 popq %r10
183 popq %r9
184 popq %r8
185 popq %rcx
186 popq %rdx
187 popq %rsi
188 popq %rdi
189 popq %rax
190 movq -48(%rbp), %rdx
191 movq %rdx, -96(%rbp)
192 pushq %rax
193 pushq %rdi
194 pushq %rsi
195 pushq %rdx
196 pushq %rcx
197 pushq %r8
198 pushq %r9
199 pushq %r10
200 pushq %r11
201 pushq -96(%rbp)
202 call Dog.speak
203 addq $8, %rsp
204 movq %rax, -112(%rbp)
205 popq %r11
206 popq %r10
207 popq %r9
208 popq %r8
209 popq %rcx
210 popq %rdx
211 popq %rsi
212 popq %rdi
213 popq %rax
214 movq $60, %rax
215 xorq %rdi, %rdi
216 syscall
217 global .print:
218     pushq %rbp
219     movq %rsp, %rbp
220     pushq %rbx
221     pushq %r12
222     pushq %r13
223     pushq %r14
224     pushq %r15
225
226     # Ensure stack alignment for printf
227     movq %rsp, %rax      # Store current rsp
228     andq $15, %rax      # Check lower 4 bits,
229     # rsp should be 16-byte aligned before call
230     jz align_ok          # If zero, alignment is
231     # okay
232     subq $8, %rsp        # Adjust stack to be
233     # 16-byte aligned
234     movq $1, -192(%rbp)  # Mark that we
235     # adjusted the stack
236
237 align_ok:
238     # Check type in %rdi (1=string, otherwise integer)
239     cmpq $0, %rdi
240     je print_string
241
242     # Setup for printing integer
243     leaq integer_format(%rip), %rdi # Load format
244     # string for integer
245     movq 16(%rbp), %rsi             # Load integer
246     # value from stack
247     xor %rax, %rax                 # Float register
248     # count
249     call printf                   # Call printf
250     jmp cleanup                   # Jump to
251     # cleanup

```

```

245 print_string:
246     # Setup for printing string
247     leaq string_format(%rip), %rdi # Load format
248     # string for string
249     movq 16(%rbp), %rsi             # Load string
250     # pointer from stack
251     xor %rax, %rax                 # Float register
252     # count
253     call printf                   # Call printf
254
255 cleanup:
256     # Restore the stack if it was misaligned
257     movb -192(%rbp), %al           # Check if we
258     # marked the stack as adjusted
259     testb $1, %al
260     jz stack_ok
261     addq $8, %rsp                 # Restore
262     # original stack alignment
263
264 stack_ok:
265     popq %r15
266     popq %r14
267     popq %r13
268     popq %r12
269     popq %rbx
270     popq %rbp
271     ret
272
273 memalloc:
274     pushq %rbp
275     mov %rsp, %rbp
276     pushq %rbx
277     pushq %rdi
278     pushq %rsi
279     pushq %r12
280     pushq %r13
281     pushq %r14
282     pushq %r15
283
284     testq $15, %rsp
285     jz is_mem_aligned
286
287     pushq $0                      # align to 16 bytes
288
289     movq 16(%rbp), %rdi
290     call malloc
291
292     add $8, %rsp                  # remove padding
293
294     jmp mem_done
295
296 is_mem_aligned:
297     movq 16(%rbp), %rdi
298     call malloc
299
300 mem_done:
301     popq %r15
302     popq %r14
303     popq %r13
304     popq %r12
305     popq %rsi
306     popq %rdi
307     popq %rbx
308     popq %rbp
309     ret

```

Code 2. The corresponding x86 Assembly of Example 1.

## Output of Example 1

woof

## Example 2

```

1 def main() -> None:
2     arr: list[int] = [10, 7, 8, 9, 1, 5]
3     arr[2]=3
4     arr[4]=arr[0]
5     i:int=0
6     for i in range(len(arr)):
7         print(arr[i])
8

```

```

9  if __name__ == "__main__":
10     main()

```

### Code 3. Arrays along with their referencing and dereferencing

```

1  .data
2      integer_format: .asciz "%d\n"
3      string_format: .asciz "%s\n"
4      .str0:
5          .string "__main__"
6      .global main
7      .text
8  main:
9      pushq %rbp
10     movq %rsp, %rbp
11     pushq %rbx
12     pushq %r12
13     pushq %r13
14     pushq %r14
15     pushq %r15
16     subq $224, %rsp
17     pushq %rax
18     pushq %rdi
19     pushq %rsi
20     pushq %rdx
21     pushq %rcx
22     pushq %r8
23     pushq %r9
24     pushq %r10
25     pushq %r11
26     pushq $48
27     call memalloc
28     add $8, %rsp
29     movq %rax, -48(%rbp)
30     popq %r11
31     popq %r10
32     popq %r9
33     popq %r8
34     popq %rcx
35     popq %rdx
36     popq %rsi
37     popq %rdi
38     popq %rax
39     movq -48(%rbp), %rax
40     movq $10, %rdx
41     movq %rdx, (%rax)
42     movq -48(%rbp), %rdx
43     add $8, %rdx
44     movq %rdx, -48(%rbp)
45     movq -48(%rbp), %rax
46     movq $7, %rdx
47     movq %rdx, (%rax)
48     movq -48(%rbp), %rdx
49     add $8, %rdx
50     movq %rdx, -48(%rbp)
51     movq -48(%rbp), %rax
52     movq $8, %rdx
53     movq %rdx, (%rax)
54     movq -48(%rbp), %rdx
55     add $8, %rdx
56     movq %rdx, -48(%rbp)
57     movq -48(%rbp), %rax
58     movq $9, %rdx
59     movq %rdx, (%rax)
60     movq -48(%rbp), %rdx
61     add $8, %rdx
62     movq %rdx, -48(%rbp)
63     movq -48(%rbp), %rax
64     movq $1, %rdx
65     movq %rdx, (%rax)
66     movq -48(%rbp), %rdx
67     add $8, %rdx
68     movq %rdx, -48(%rbp)
69     movq -48(%rbp), %rax
70     movq $5, %rdx
71     movq %rdx, (%rax)
72     movq -48(%rbp), %rdx
73     add $8, %rdx
74     movq %rdx, -48(%rbp)
75     movq -48(%rbp), %rdx
76     sub $48, %rdx
77     movq %rdx, -48(%rbp)
78     movq -48(%rbp), %rdx
79     movq %rdx, -64(%rbp)
80     movq $2, %rdx
81     imul $8, %rdx
82     movq %rdx, -72(%rbp)
83     movq -64(%rbp), %rdx

```

```

84     add -72(%rbp), %rdx
85     movq %rdx, -72(%rbp)
86     movq -72(%rbp), %rax
87     movq $3, %rdx
88     movq %rdx, (%rax)
89     movq $4, %rdx
90     imul $8, %rdx
91     movq %rdx, -88(%rbp)
92     movq -64(%rbp), %rdx
93     add -88(%rbp), %rdx
94     movq %rdx, -88(%rbp)
95     movq $0, %rdx
96     imul $8, %rdx
97     movq %rdx, -96(%rbp)
98     movq -64(%rbp), %rdx
99     add -96(%rbp), %rdx
100    movq %rdx, -96(%rbp)
101    movq -96(%rbp), %rax
102    movq (%rax), %rdx
103    movq %rdx, -112(%rbp)
104    movq -88(%rbp), %rax
105    movq -112(%rbp), %rdx
106    movq %rdx, (%rax)
107    movq $0, -128(%rbp)
108    movq $6, -136(%rbp)
109    movq $0, -144(%rbp)
110 L2:
111    movq -144(%rbp), %rdx
112    movq %rdx, -128(%rbp)
113    movq -144(%rbp), %rbx
114    movq -136(%rbp), %rcx
115    cmp %rcx, %rbx
116    jl L4
117    jmp L1
118 L4:
119    movq -128(%rbp), %rdx
120    imul $8, %rdx
121    movq %rdx, -184(%rbp)
122    movq -64(%rbp), %rdx
123    add -184(%rbp), %rdx
124    movq %rdx, -184(%rbp)
125    movq -184(%rbp), %rax
126    movq (%rax), %rdx
127    movq %rdx, -200(%rbp)
128    pushq %rax
129    pushq %rdi
130    pushq %rsi
131    pushq %rdx
132    pushq %rcx
133    pushq %r8
134    pushq %r9
135    pushq %r10
136    pushq %r11
137    pushq -200(%rbp)
138    call global.print
139    addq $8, %rsp
140    popq %r11
141    popq %r10
142    popq %r9
143    popq %r8
144    popq %rcx
145    popq %rdx
146    popq %rsi
147    popq %rdi
148    popq %rax
149 L3:
150    movq -144(%rbp), %rdx
151    add $1, %rdx
152    movq %rdx, -144(%rbp)
153    jmp L2
154 L1:
155    movq $60, %rax
156    xorq %rdi, %rdi
157    syscall
158 global.print:
159     pushq %rbp
160     movq %rsp, %rbp
161     pushq %rbx
162     pushq %r12
163     pushq %r13
164     pushq %r14
165     pushq %r15
166
167     # Ensure stack alignment for printf
168     movq %rsp, %rax      # Store current rsp
169     andq $15, %rax      # Check lower 4 bits,
170                          # rsp should be 16-byte aligned before call
171     jz    align_ok      # If zero, alignment is
                          # okay
172     subq $8, %rsp      # Adjust stack to be
                          # 16-byte aligned

```

```

172     movq    $1, -192(%rbp)      # Mark that we
    adjusted the stack
173
174 align_ok:
175     # Check type in %rdi (1=string, otherwise integer)
176     cmpq    $0, %rdi
177     je      print_string
178
179     # Setup for printing integer
180
181     leaq    integer_format(%rip), %rdi # Load format
    string for integer
182     movq    16(%rbp), %rsi      # Load integer
    value from stack
183     xor     %rax, %rax          # Float register
    count
184     call    printf              # Call printf
185     jmp     cleanup            # Jump to
    cleanup
186
187 print_string:
188     # Setup for printing string
189     leaq    string_format(%rip), %rdi # Load format
    string for string
190     movq    16(%rbp), %rsi      # Load string
    pointer from stack
191     xor     %rax, %rax          # Float register
    count
192     call    printf              # Call printf
193
194 cleanup:
195     # Restore the stack if it was misaligned
196     movb    -192(%rbp), %al      # Check if we
    marked the stack as adjusted
197     testb   $1, %al
198     jz      stack_ok
199     addq    $8, %rsp            # Restore
    original stack alignment
200
201 stack_ok:
202     popq    %r15
203     popq    %r14
204     popq    %r13
205     popq    %r12
206     popq    %rbx
207     popq    %rbp
208     ret
209 memalloc:
210     pushq   %rbp
211     mov     %rsp, %rbp
212     pushq   %rbx
213     pushq   %rdi
214     pushq   %rsi
215     pushq   %r12
216     pushq   %r13
217     pushq   %r14
218     pushq   %r15
219
220     testq   $15, %rsp
221     jz      is_mem_aligned
222
223     pushq   $0                  # align to 16 bytes
224
225     movq    16(%rbp), %rdi
226     call    malloc
227
228     add     $8, %rsp            # remove padding
229
230     jmp     mem_done
231
232 is_mem_aligned:
233
234     movq    16(%rbp), %rdi
235     call    malloc
236
237 mem_done:
238
239     popq    %r15
240     popq    %r14
241     popq    %r13
242     popq    %r12
243     popq    %rsi
244     popq    %rdi
245     popq    %rbx
246     popq    %rbp
247
248     ret

```

Code 4. The corresponding x86 Assembly of Example 2.

## Output of Example 2

```

10
7
3
9
10
5

```

## Example 3

```

1 def string_comparison(string1: str, string2: str, n1:int
    , n2:int) -> sNone:
2     if string1 == string2:
3         print("The strings are equal.")
4     else:
5         print("String are not equal.")
6
7 def main():
8     string_comparison("apple", "banana",5,6)
9     string_comparison("python", "python",6,6)
10
11 if __name__ == "__main__":
12     main()

```

Code 5. Strings and their comparison

```

1 .data
2     integer_format: .asciz "%d\n"
3     string_format: .asciz "%s\n"
4     .str0:
5         .string "The strings are equal."
6     .str1:
7         .string "String are not equal."
8     .str2:
9         .string "apple"
10    .str3:
11        .string "banana"
12    .str4:
13        .string "python"
14    .str5:
15        .string "python"
16    .str6:
17        .string "__main__"
18    .global main
19 .text
20 global string_comparison:
21     pushq   %rbp
22     movq    %rsp, %rbp
23     pushq   %rbx
24     pushq   %r12
25     pushq   %r13
26     pushq   %r14
27     pushq   %r15
28     subq    $120, %rsp
29     movq    16(%rbp), %rbx
30     movq    24(%rbp), %rcx
31     movq    %rbx, %rdi
32     movq    %rcx, %rsi
33     call    strcmp
34     movsx   %eax, %rax
35     cmp     $0, %rax
36     je      L2
37     jmp     L3
38 L2:
39     lea     .str0(%rip), %rax
40     movq    %rax, -80(%rbp)
41     movq    $0, %rdi
42     pushq   %rax
43     pushq   %rdi
44     pushq   %rsi
45     pushq   %rdx
46     pushq   %rcx
47     pushq   %r8
48     pushq   %r9
49     pushq   %r10
50     pushq   %r11
51     pushq   -80(%rbp)
52     call    global.print
53     addq    $8, %rsp
54     popq    %r11
55     popq    %r10
56     popq    %r9

```

```

57 popq %r8
58 popq %rcx
59 popq %rdx
60 popq %rsi
61 popq %rdi
62 popq %rax
63 movq $1, %rdi
64 jmp L1
65 L3:
66 lea .str1(%rip), %rax
67 movq %rax, -112(%rbp)
68 movq $0, %rdi
69 pushq %rax
70 pushq %rdi
71 pushq %rsi
72 pushq %rdx
73 pushq %rcx
74 pushq %r8
75 pushq %r9
76 pushq %r10
77 pushq %r11
78 pushq -112(%rbp)
79 call global.print
80 addq $8, %rsp
81 popq %r11
82 popq %r10
83 popq %r9
84 popq %r8
85 popq %rcx
86 popq %rdx
87 popq %rsi
88 popq %rdi
89 popq %rax
90 movq $1, %rdi
91 L1:
92 addq $120, %rsp
93 popq %r15
94 popq %r14
95 popq %r13
96 popq %r12
97 popq %rbx
98 popq %rbp
99 ret
100 main:
101 pushq %rbp
102 movq %rsp, %rbp
103 pushq %rbx
104 pushq %r12
105 pushq %r13
106 pushq %r14
107 pushq %r15
108 subq $120, %rsp
109 lea .str3(%rip), %rax
110 movq %rax, -48(%rbp)
111 lea .str2(%rip), %rax
112 movq %rax, -64(%rbp)
113 pushq %rax
114 pushq %rdi
115 pushq %rsi
116 pushq %rdx
117 pushq %rcx
118 pushq %r8
119 pushq %r9
120 pushq %r10
121 pushq %r11
122 pushq $6
123 pushq $5
124 pushq -48(%rbp)
125 pushq -64(%rbp)
126 call global.string_comparison
127 addq $32, %rsp
128 movq %rax, -80(%rbp)
129 popq %r11
130 popq %r10
131 popq %r9
132 popq %r8
133 popq %rcx
134 popq %rdx
135 popq %rsi
136 popq %rdi
137 popq %rax
138 movq $1, %rdi
139 lea .str5(%rip), %rax
140 movq %rax, -96(%rbp)
141 lea .str5(%rip), %rax
142 movq %rax, -104(%rbp)
143 pushq %rax
144 pushq %rdi
145 pushq %rsi
146 pushq %rdx
147 pushq %rcx

```

```

148 pushq %r8
149 pushq %r9
150 pushq %r10
151 pushq %r11
152 pushq $6
153 pushq $6
154 pushq -96(%rbp)
155 pushq -104(%rbp)
156 call global.string_comparison
157 addq $32, %rsp
158 movq %rax, -112(%rbp)
159 popq %r11
160 popq %r10
161 popq %r9
162 popq %r8
163 popq %rcx
164 popq %rdx
165 popq %rsi
166 popq %rdi
167 popq %rax
168 movq $1, %rdi
169 movq $60, %rax
170 xorq %rdi, %rdi
171 syscall
172 global.print:
173     pushq %rbp
174     movq %rsp, %rbp
175     pushq %rbx
176     pushq %r12
177     pushq %r13
178     pushq %r14
179     pushq %r15
180
181     # Ensure stack alignment for printf
182     movq %rsp, %rax      # Store current rsp
183     andq $15, %rax      # Check lower 4 bits,
                          # rsp should be 16-byte aligned before call
184     jz align_ok         # If zero, alignment is
                          # okay
185     subq $8, %rsp        # Adjust stack to be
                          # 16-byte aligned
186     movq $1, -192(%rbp)  # Mark that we
                          # adjusted the stack
187
188 align_ok:
189     # Check type in %rdi (1=string, otherwise integer)
190     cmpq $0, %rdi
191     je print_string
192
193     # Setup for printing integer
194
195     leaq integer_format(%rip), %rdi # Load format
196     string for integer
197     movq 16(%rbp), %rsi             # Load integer
198     value from stack
199     xor %rax, %rax                 # Float register
200     count
201     call printf                    # Call printf
202     jmp cleanup                    # Jump to
203     cleanup
204
205 print_string:
206     # Setup for printing string
207     leaq string_format(%rip), %rdi # Load format
208     string for string
209     movq 16(%rbp), %rsi             # Load string
210     pointer from stack
211     xor %rax, %rax                 # Float register
212     count
213     call printf                    # Call printf
214
215 cleanup:
216     # Restore the stack if it was misaligned
217     movb -192(%rbp), %al           # Check if we
218     marked the stack as adjusted
219     testb $1, %al
220     jz stack_ok
221     addq $8, %rsp                  # Restore
222     original stack alignment
223
224 stack_ok:
225     popq %r15
226     popq %r14
227     popq %r13
228     popq %r12
229     popq %rbx
230     popq %rbp
231     ret
232
233 memalloc:
234     pushq %rbp
235     mov %rsp, %rbp

```

```

226     pushq %rbx
227     pushq %rdi
228     pushq %rsi
229     pushq %r12
230     pushq %r13
231     pushq %r14
232     pushq %r15
233
234     testq $15, %rsp
235     jz is_mem_aligned
236
237     pushq $0                # align to 16 bytes
238
239     movq 16(%rbp), %rdi
240     call malloc
241
242     add $8, %rsp            # remove padding
243
244     jmp mem_done
245
246 is_mem_aligned:
247
248     movq 16(%rbp), %rdi
249     call malloc
250
251 mem_done:
252
253     popq %r15
254     popq %r14
255     popq %r13
256     popq %r12
257     popq %rsi
258     popq %rdi
259     popq %rbx
260     popq %rbp
261
262     ret

```

Code 6. The corresponding x86 Assembly of Example 3

### Output of Example 3

```

String are not equal.
The strings are equal.

```

### Example 4

```

1 def main() -> None:
2     x: int = 5
3     y: int = 3
4
5     print(x+y)
6     print(x & y)
7     print(x-(y*(x//y)))
8     print(x ^ y)
9     print(x | (x&y))
10
11 if __name__ == "__main__":
12     main()

```

Code 7. Evaluation of expressions inside strings similarly function calls can also be done inside print.

```

1 .data
2     integer_format: .asciz "%d\n"
3     string_format: .asciz "%s\n"
4     .str0:
5         .string "__main__"
6     .global main
7     .text
8     main:
9         pushq %rbp
10        movq %rsp, %rbp
11        pushq %rbx
12        pushq %r12
13        pushq %r13
14        pushq %r14
15        pushq %r15
16        subq $128, %rsp
17        movq $5, -40(%rbp)
18        movq $3, -48(%rbp)
19        movq -40(%rbp), %rdx
20        add -48(%rbp), %rdx

```

```

21     movq %rdx, -56(%rbp)
22     pushq %rax
23     pushq %rdi
24     pushq %rsi
25     pushq %rdx
26     pushq %rcx
27     pushq %r8
28     pushq %r9
29     pushq %r10
30     pushq %r11
31     pushq -56(%rbp)
32     call global.print
33     addq $8, %rsp
34     popq %r11
35     popq %r10
36     popq %r9
37     popq %r8
38     popq %rcx
39     popq %rdx
40     popq %rsi
41     popq %rdi
42     popq %rax
43     movq -40(%rbp), %rax
44     movq -48(%rbp), %rcx
45     and %rcx, %rax
46     movq %rax, -72(%rbp)
47     pushq %rax
48     pushq %rdi
49     pushq %rsi
50     pushq %rdx
51     pushq %rcx
52     pushq %r8
53     pushq %r9
54     pushq %r10
55     pushq %r11
56     pushq -72(%rbp)
57     call global.print
58     addq $8, %rsp
59     popq %r11
60     popq %r10
61     popq %r9
62     popq %r8
63     popq %rcx
64     popq %rdx
65     popq %rsi
66     popq %rdi
67     popq %rax
68     movq -40(%rbp), %rax
69     cqto
70     movq -48(%rbp), %rcx
71     idiv %rcx
72     movq %rax, -80(%rbp)
73     movq -48(%rbp), %rdx
74     imul -80(%rbp), %rdx
75     movq %rdx, -88(%rbp)
76     movq -40(%rbp), %rdx
77     sub -88(%rbp), %rdx
78     movq %rdx, -96(%rbp)
79     pushq %rax
80     pushq %rdi
81     pushq %rsi
82     pushq %rdx
83     pushq %rcx
84     pushq %r8
85     pushq %r9
86     pushq %r10
87     pushq %r11
88     pushq -96(%rbp)
89     call global.print
90     addq $8, %rsp
91     popq %r11
92     popq %r10
93     popq %r9
94     popq %r8
95     popq %rcx
96     popq %rdx
97     popq %rsi
98     popq %rdi
99     popq %rax
100    movq -40(%rbp), %rax
101    movq -48(%rbp), %rcx
102    xor %rcx, %rax
103    movq %rax, -104(%rbp)
104    pushq %rax
105    pushq %rdi
106    pushq %rsi
107    pushq %rdx
108    pushq %rcx
109    pushq %r8
110    pushq %r9
111    pushq %r10

```



```

112 pushq %r11
113 pushq -104(%rbp)
114 call global.print
115 addq $8, %rsp
116 popq %r11
117 popq %r10
118 popq %r9
119 popq %r8
120 popq %rcx
121 popq %rdx
122 popq %rsi
123 popq %rdi
124 popq %rax
125 movq -40(%rbp), %rax
126 movq -48(%rbp), %rcx
127 and %rcx, %rax
128 movq %rax, -112(%rbp)
129 movq -40(%rbp), %rax
130 movq -112(%rbp), %rcx
131 or %rcx, %rax
132 movq %rax, -120(%rbp)
133 pushq %rax
134 pushq %rdi
135 pushq %rsi
136 pushq %rdx
137 pushq %rcx
138 pushq %r8
139 pushq %r9
140 pushq %r10
141 pushq %r11
142 pushq -120(%rbp)
143 call global.print
144 addq $8, %rsp
145 popq %r11
146 popq %r10
147 popq %r9
148 popq %r8
149 popq %rcx
150 popq %rdx
151 popq %rsi
152 popq %rdi
153 popq %rax
154 movq $60, %rax
155 xorq %rdi, %rdi
156 syscall
157 global.print:
158     pushq %rbp
159     movq %rsp, %rbp
160     pushq %rbx
161     pushq %r12
162     pushq %r13
163     pushq %r14
164     pushq %r15
165
166     # Ensure stack alignment for printf
167     movq %rsp, %rax          # Store current rsp
168     andq $15, %rax          # Check lower 4 bits,
169     # rsp should be 16-byte aligned before call
170     jz align_ok              # If zero, alignment is
171     # okay
172     subq $8, %rsp            # Adjust stack to be
173     # 16-byte aligned
174     movq $1, -192(%rbp)      # Mark that we
175     # adjusted the stack
176
177 align_ok:
178     # Check type in %rdi (1=string, otherwise integer)
179     cmpq $0, %rdi
180     je print_string
181
182     # Setup for printing integer
183     leaq integer_format(%rip), %rdi # Load format
184     # string for integer
185     movq 16(%rbp), %rsi        # Load integer
186     # value from stack
187     xor %rax, %rax            # Float register
188     # count
189     call printf               # Call printf
190     jmp cleanup               # Jump to
191     # cleanup
192
193 print_string:
194     # Setup for printing string
195     leaq string_format(%rip), %rdi # Load format
196     # string for string
197     movq 16(%rbp), %rsi        # Load string
198     # pointer from stack
199     xor %rax, %rax            # Float register
200     # count
201     call printf               # Call printf

```

```

192 cleanup:
193     # Restore the stack if it was misaligned
194     movb -192(%rbp), %al      # Check if we
195     # marked the stack as adjusted
196     testb $1, %al
197     jz stack_ok
198     addq $8, %rsp            # Restore
199     # original stack alignment
200
201 stack_ok:
202     popq %r15
203     popq %r14
204     popq %r13
205     popq %r12
206     popq %rbx
207     popq %rbp
208     ret
209
210 memalloc:
211     pushq %rbp
212     mov %rsp, %rbp
213     pushq %rbx
214     pushq %rdi
215     pushq %rsi
216     pushq %r12
217     pushq %r13
218     pushq %r14
219     pushq %r15
220
221     testq $15, %rsp
222     jz is_mem_aligned
223
224     pushq $0                  # align to 16 bytes
225
226     movq 16(%rbp), %rdi
227     call malloc
228
229     add $8, %rsp              # remove padding
230
231     jmp mem_done
232
233 is_mem_aligned:
234     movq 16(%rbp), %rdi
235     call malloc
236
237 mem_done:
238     popq %r15
239     popq %r14
240     popq %r13
241     popq %r12
242     popq %rsi
243     popq %rdi
244     popq %rbx
245     popq %rbp
246
247     ret

```

Code 8. The corresponding x86 Assembly of Example 4

## Output of Example 4

```

8
1
2
6
5

```

## Example 5

```

1 def main() -> None:
2     # no of identical pairs
3     arr: list[int]=[1,2,3,1,2]
4     x: int = 0
5     y: int = 0
6     sol:int =0
7     while x < len(arr):
8         for y in range(len(arr)-x-1):
9             if arr[x]==arr[y+1+x]:
10                 sol=sol+1
11         x=x+1
12     print(sol)
13

```



```

14 if __name__ == "__main__":
15     main()

```

Code 9. Strings and their comparison

```

1 .data
2     integer_format: .asciz "%d\n"
3     string_format: .asciz "%s\n"
4     .str0:
5         .string "__main__"
6     .global main
7     .text
8 main:
9     pushq %rbp
10    movq %rsp, %rbp
11    pushq %rbx
12    pushq %r12
13    pushq %r13
14    pushq %r14
15    pushq %r15
16    subq $320, %rsp
17    pushq %rax
18    pushq %rdi
19    pushq %rsi
20    pushq %rdx
21    pushq %rcx
22    pushq %r8
23    pushq %r9
24    pushq %r10
25    pushq %r11
26    pushq $40
27    call memalloc
28    add $8, %rsp
29    movq %rax, -48(%rbp)
30    popq %r11
31    popq %r10
32    popq %r9
33    popq %r8
34    popq %rcx
35    popq %rdx
36    popq %rsi
37    popq %rdi
38    popq %rax
39    movq -48(%rbp), %rax
40    movq $1, %rdx
41    movq %rdx, (%rax)
42    movq -48(%rbp), %rdx
43    add $8, %rdx
44    movq %rdx, -48(%rbp)
45    movq -48(%rbp), %rax
46    movq $2, %rdx
47    movq %rdx, (%rax)
48    movq -48(%rbp), %rdx
49    add $8, %rdx
50    movq %rdx, -48(%rbp)
51    movq -48(%rbp), %rax
52    movq $3, %rdx
53    movq %rdx, (%rax)
54    movq -48(%rbp), %rdx
55    add $8, %rdx
56    movq %rdx, -48(%rbp)
57    movq -48(%rbp), %rax
58    movq $1, %rdx
59    movq %rdx, (%rax)
60    movq -48(%rbp), %rdx
61    add $8, %rdx
62    movq %rdx, -48(%rbp)
63    movq -48(%rbp), %rax
64    movq $2, %rdx
65    movq %rdx, (%rax)
66    movq -48(%rbp), %rdx
67    add $8, %rdx
68    movq %rdx, -48(%rbp)
69    movq -48(%rbp), %rdx
70    sub $40, %rdx
71    movq %rdx, -48(%rbp)
72    movq -48(%rbp), %rdx
73    movq %rdx, -64(%rbp)
74    movq $0, -72(%rbp)
75    movq $0, -80(%rbp)
76    movq $0, -88(%rbp)
77 L2:
78    movq $5, -96(%rbp)
79    movq -72(%rbp), %rbx
80    movq -96(%rbp), %rcx
81    cmp %rcx, %rbx
82    jl L9
83    jmp L1

```

```

84 L9:
85    movq $5, -136(%rbp)
86    movq -136(%rbp), %rdx
87    sub -72(%rbp), %rdx
88    movq %rdx, -144(%rbp)
89    movq -144(%rbp), %rdx
90    sub $1, %rdx
91    movq %rdx, -152(%rbp)
92    movq $0, -160(%rbp)
93 L4:
94    movq -160(%rbp), %rdx
95    movq %rdx, -80(%rbp)
96    movq -160(%rbp), %rbx
97    movq -152(%rbp), %rcx
98    cmp %rcx, %rbx
99    jl L8
100   jmp L3
101 L8:
102    movq -72(%rbp), %rdx
103    imul $8, %rdx
104    movq %rdx, -192(%rbp)
105    movq -64(%rbp), %rdx
106    add -192(%rbp), %rdx
107    movq %rdx, -192(%rbp)
108    movq -192(%rbp), %rax
109    movq (%rax), %rdx
110    movq %rdx, -208(%rbp)
111    movq -80(%rbp), %rdx
112    add $1, %rdx
113    movq %rdx, -216(%rbp)
114    movq -216(%rbp), %rdx
115    add -72(%rbp), %rdx
116    movq %rdx, -224(%rbp)
117    movq -224(%rbp), %rdx
118    imul $8, %rdx
119    movq %rdx, -232(%rbp)
120    movq -64(%rbp), %rdx
121    add -232(%rbp), %rdx
122    movq %rdx, -232(%rbp)
123    movq -232(%rbp), %rax
124    movq (%rax), %rdx
125    movq %rdx, -248(%rbp)
126    movq -208(%rbp), %rbx
127    movq -248(%rbp), %rcx
128    cmp %rcx, %rbx
129    je L7
130    jmp L6
131 L7:
132    movq -88(%rbp), %rdx
133    add $1, %rdx
134    movq %rdx, -280(%rbp)
135    movq -280(%rbp), %rdx
136    movq %rdx, -88(%rbp)
137    jmp L6
138 L6:
139 L5:
140    movq -160(%rbp), %rdx
141    add $1, %rdx
142    movq %rdx, -160(%rbp)
143    jmp L4
144 L3:
145    movq -72(%rbp), %rdx
146    add $1, %rdx
147    movq %rdx, -296(%rbp)
148    movq -296(%rbp), %rdx
149    movq %rdx, -72(%rbp)
150    jmp L2
151 L1:
152    pushq %rax
153    pushq %rdi
154    pushq %rsi
155    pushq %rdx
156    pushq %rcx
157    pushq %r8
158    pushq %r9
159    pushq %r10
160    pushq %r11
161    pushq -88(%rbp)
162    call global.print
163    addq $8, %rsp
164    popq %r11
165    popq %r10
166    popq %r9
167    popq %r8
168    popq %rcx
169    popq %rdx
170    popq %rsi
171    popq %rdi
172    popq %rax
173    movq $60, %rax
174    xorq %rdi, %rdi

```

```

175     syscall
176 global .print:
177     pushq   %rbp
178     movq    %rsp, %rbp
179     pushq   %rbx
180     pushq   %r12
181     pushq   %r13
182     pushq   %r14
183     pushq   %r15
184
185     # Ensure stack alignment for printf
186     movq    %rsp, %rax      # Store current rsp
187     andq    $15, %rax      # Check lower 4 bits,
188     # rsp should be 16-byte aligned before call
189     jz      align_ok       # If zero, alignment is
190     # okay
191     subq    $8, %rsp        # Adjust stack to be
192     # 16-byte aligned
193     movq    $1, -192(%rbp)  # Mark that we
194     # adjusted the stack
195
196 align_ok:
197     # Check type in %rdi (1=string, otherwise integer)
198     cmpq    $0, %rdi
199     je      print_string
200
201     # Setup for printing integer
202     leaq    integer_format(%rip), %rdi # Load format
203     # string for integer
204     movq    16(%rbp), %rsi          # Load integer
205     # value from stack
206     xor     %rax, %rax              # Float register
207     # count
208     call    printf                 # Call printf
209     jmp     cleanup                # Jump to
210     # cleanup
211
212 print_string:
213     # Setup for printing string
214     leaq    string_format(%rip), %rdi # Load format
215     # string for string
216     movq    16(%rbp), %rsi          # Load string
217     # pointer from stack
218     xor     %rax, %rax              # Float register
219     # count
220     call    printf                 # Call printf
221
222 cleanup:
223     # Restore the stack if it was misaligned
224     movb    -192(%rbp), %al         # Check if we
225     # marked the stack as adjusted
226     testb   $1, %al
227     jz      stack_ok
228     addq    $8, %rsp                # Restore
229     # original stack alignment
230
231 stack_ok:
232     popq    %r15
233     popq    %r14
234     popq    %r13
235     popq    %r12
236     popq    %rbx
237     popq    %rbp
238     ret
239
240 memalloc:
241     pushq   %rbp
242     mov     %rsp, %rbp
243     pushq   %rbx
244     pushq   %rdi
245     pushq   %rsi
246     pushq   %r12
247     pushq   %r13
248     pushq   %r14
249     pushq   %r15
250
251     testq   $15, %rsp
252     jz      is_mem_aligned
253
254     pushq   $0                    # align to 16 bytes
255
256     movq    16(%rbp), %rdi
257     call    malloc
258
259     add     $8, %rsp              # remove padding
260
261     jmp     mem_done
262
263 is_mem_aligned:
264     movq    16(%rbp), %rdi

```

```

253     call    malloc
254
255 mem_done:
256
257     popq    %r15
258     popq    %r14
259     popq    %r13
260     popq    %r12
261     popq    %rsi
262     popq    %rdi
263     popq    %rbx
264     popq    %rbp
265
266     ret

```

**Code 10.** The usage of `len` inside `range` for usage in a loop. Along with some array handling

### Output of Example 5

2

### Effort Log

All the team members have contributed equally to the project.