Milestone 3

Indian Institute of Technology Kanpur

Semester II, 2023-24

Course: CS335
Group No. 3
Team Members:

- Manasvi Jain - 210581
- Sarthak Kalankar - 210935
- Saugat Kannojia - 210943

Contents

1 Compilation and Execution Instructions **2**

1.1 Setup 2

1.2 Running the shell script 2

2 Assumptions **2**

3 Language Features **3**

4 x86 Implemenation **4**

5 Error Handling **14**

1 Compilation and Execution Instructions

1.1 Setup

1. Extract or clone the main folder where all the source files are inside 'milestone3' directory.
2. Download the following libraries in milestone3 directory: bison, flex.

```
sudo apt-get update
sudo apt-get install flex
sudo apt-get install bison
```

1.2 Running the shell script

1. Run the wrapper shell script 'x86_simu.sh' in **milestone3** directory by the following commands (First command to be executed if you encounter an error while executing the script) :

```
sed -i 's/\r$//' src/main_script.sh
sed -i 's/\r$//' src/x86_simu.sh
./src/x86_simu.sh tests/<testcasename>.py
```

2 Assumptions

- We expect only one value to be returned from a function.
- We have not considered unary minus operator as a separate operation, and thus we have combined that with one other operation, if applicable.
- We assume that type casting shouldn't happen for an example case when the function argument expects float, and we provide a variable which is int inside the procedure call.
- We make the assumption that all the variable will be declared with a type.
- We cannot call a function within a function, which also discards operations like range(len(array)).
- Range in for loop can include upto 3 arguments, where they are single values or variable but not an expression.
- Since, we are not using any float point registers, it is assumed that there will be no float in the entire program and hence no implicit type casting for $int \longleftrightarrow float$.
- For any class, all the self attributes can only be defined in the *__init__* method of the class and "self" would always be placed as the first attribute in all methods declared inside the class.

3 Language Features

Classes

- Can create a class object and access it's attributes and methods
- While inheriting a parent class, all of the parent class's self attribute as well as methods can be accessed here
- Can support multilevel inheritance and constructors

Functions

- Can call multiple functions inside definiton
- Function calls with recursion
- Can pass arrays along with other data types in function
- Supports Range function with 1,2 or 3 arguments
- Supports Len function with 1 argument which should be a list
- Supports Print function with argument which can either be a string or a variable

Array Referencing

- Can pass arrays in functions
- All the elements of an array can be updated if needed
- Array index can only be an integer/variable of type int

Miscellaneous

- String comparison is supported, returns 0/1 according to the operations
- Typecasting: True and False are converted to 1 and 0 respectively
- All the basic operators are supported
- Control flow via if-elif-else, for, while, break and continue
- Logical Operators have been handled.

4 x86 Implemenation

For brevity, for each x86 code we havent added memalloc and print everywhere but just separately at the end.

Expressions

Python Code

```python
def main():
    a : int = 10
    b : int = 20
    c : int = 3

    x:int = a + b * c

if __name__ == "__main__":
    main()
```

x86 Code

```
    .section        .rodata
.str0:
    .string "__main__"
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movq    $10, -8(%rbp)
    subq    $8, %rsp
    movq    $20, -16(%rbp)
    subq    $8, %rsp
    movq    $3, -24(%rbp)
    subq    $8, %rsp
    movq    -16(%rbp), %rax
    movq    -24(%rbp), %r8
    imul    %r8, %rax
    movq    %rax, -32(%rbp)
    subq    $8, %rsp
    movq    -8(%rbp), %r12
    movq    -32(%rbp), %r8
    addq    %r8, %r12
    movq    %r12, -40(%rbp)
    leave
    ret

    subq     $8, %rsp
    movq     0(%rbp), %r12
    movq     0(%rbp), %r8
    cmp      %r8, %r12
    sete     %cl
    movzbl   %cl, %ecx
    movq     %rcx, -48(%rbp)
    movq     $1, %r8
    movq     %rcx, %r15
    and      %rcx, %r8
    jz       .L1
.L1:
.L0:
    movq     $1, %r8
    movq     %r15, %rcx
    and       %r8, %rcx
    jnz      .rsp0
    movq     $0, %r14
    sub      %r14, %rsp
.rsp0:
```

Passing Strings

Python Code

```
def main():
  i: str = "hello"
  print(i)
```

x86 Code

```
    .section        .rodata
.str0:
    .string "hello"
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    leaq    .str0(%rip), %rsi
    leaq    .LC1(%rip), %rdi
    movq    $0, %rax
    call    printf@PLT
    leave
    ret
```

Array Referencing

Python Code x86 Code

Compound Statements

For Loops

Python Code

```
def main():
  i: int = 0
  for i in range (0,10):
    print(i)
```

x86 Code

```
    .section        .rodata
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movq    $0, -8(%rbp)
    subq    $8, %rsp
    movq    $0, -16(%rbp)
    subq    $8, %rsp
    movq    $10, -24(%rbp)
    subq    $8, %rsp
    movq    $1, -32(%rbp)
    movq    -16(%rbp), %r9
    movq    %r9, -8(%rbp)
    jmp .L1
.L0:
    movq    -8(%rbp), %r9
    movq    -32(%rbp), %r8
    addq    %r8, %r9
    movq    %r9, -16(%rbp)
    movq    -16(%rbp), %r9
    movq    %r9, -8(%rbp)
.L1:
    movq    -16(%rbp), %r9
    movq    -24(%rbp), %r8
    cmp %r8, %r9
    setl    %cl
    movzbl  %cl, %ecx
    movq    $1, %r8
    and %rcx, %r8
    jz  .L2
    movq    -8(%rbp), %rsi
    leaq    .LC0(%rip), %rdi
    movq    $0, %rax
    call    printf@PLT
    jmp .L0
.L2:
    leave
    ret
```

While Loops

Python Code

```python
def main():
  i: int = 0
  j:int = 5
  while i<5:
    print(i)
    i = i + 1
```

x86 Code

```
    .section        .rodata
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movq    $0, -8(%rbp)
    subq    $8, %rsp
    movq    $5, -16(%rbp)
.L0:
    subq    $8, %rsp
    movq    $5, -24(%rbp)
    subq    $8, %rsp
    movq    -8(%rbp), %r9
    movq    -24(%rbp), %r8
    cmp %r8, %r9
    setl    %cl
    movzbl  %cl, %ecx
    movq    %rcx, -32(%rbp)
    movq    $1, %r8
    and %rcx, %r8
    jz  .L1
    movq    -8(%rbp), %rsi
    leaq    .LC0(%rip), %rdi
    movq    $0, %rax
    call    printf@PLT
    subq    $8, %rsp
    movq    $1, -40(%rbp)
    subq    $8, %rsp
    movq    -8(%rbp), %r9
    movq    -40(%rbp), %r8
    addq    %r8, %r9
    movq    %r9, -48(%rbp)
    movq    -48(%rbp), %r9
    movq    %r9, -8(%rbp)
    jmp .L0
.L1:
    leave
    ret
```

If - elif - else statements

Python Code

```
def main():
  a: str = "i < j"
  b: str = "j < i"
  c: str = "i = j"
  i:int = 6
  j:int = 5
  k: int = 3
  if(i<j):
    print(a)
  elif(j<i):
    print(b)
  else:
    print(c)
```

x86 Code

```
    .section        .rodata
.str2:
    .string "i = j"
.str1:
    .string "j < i"
.str0:
    .string "i < j"
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movq    $6, -8(%rbp)
    subq    $8, %rsp
    movq    $5, -16(%rbp)
    subq    $8, %rsp
    movq    $3, -24(%rbp)
    subq    $8, %rsp
    movq    -8(%rbp), %r9
    movq    -16(%rbp), %r8
    cmp %r8, %r9
    setl    %cl
    movzbl  %cl, %ecx
    movq    %rcx, -32(%rbp)
    movq    $1, %r8
    and %rcx, %r8
    jz  .L1
    leaq    .str0(%rip), %rsi
    leaq    .LC1(%rip), %rdi
    movq    $0, %rax
    call    printf@PLT
    jmp .L0
.L1:
    subq    $8, %rsp
    movq    -16(%rbp), %r9
    movq    -8(%rbp), %r8
    cmp %r8, %r9
    setl    %cl
    movzbl  %cl, %ecx
    movq    %rcx, -40(%rbp)
    movq    $1, %r8
    and %rcx, %r8
    jz  .L2
    leaq    .str1(%rip), %rsi
    leaq    .LC1(%rip), %rdi
    movq    $0, %rax
    subq    $8, %rsp
    call    printf@PLT
    jmp .L0
.L2:
    leaq    .str2(%rip), %rsi
    leaq    .LC1(%rip), %rdi
    movq    $0, %rax
    call    printf@PLT
.L0:
    leave
    ret
```

Functions

For any function, before calling the function, we push the argument values on stack and then access them after entering the function label. An example is show below:

Python Code

```python
def add(x: int, y: int, z:int) -> int:
  return x + y + z

def main():
  a:int = 8
  b:int = 4
  c:int = 9
  d:int = add(a,b,c)
  print(d)
```

x86 Code

```
    .section        .rodata
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  add
    .type   add, @function
add:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $24, %rsp
    movq    16(%rbp), %r9
    movq    %r9, -24(%rbp)
    movq    24(%rbp), %r9
    movq    %r9, -16(%rbp)
    movq    32(%rbp), %r9
    movq    %r9, -8(%rbp)
    subq    $8, %rsp
    movq    -8(%rbp), %r9
    movq    -16(%rbp), %r8
    addq    %r8, %r9
    movq    %r9, -32(%rbp)
    subq    $8, %rsp
    movq    -32(%rbp), %r9
    movq    -24(%rbp), %r8
    addq    %r8, %r9
    movq    %r9, -40(%rbp)
    movq    -40(%rbp), %rax
    leave
    ret

    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movq    $8, -8(%rbp)
    subq    $8, %rsp
    movq    $4, -16(%rbp)
    subq    $8, %rsp
    movq    $9, -24(%rbp)
    subq    $8, %rsp
    movq    -8(%rbp), %r8
    movq    %r8, -32(%rbp)
    subq    $8, %rsp
    movq    -16(%rbp), %r8
    movq    %r8, -40(%rbp)
    subq    $8, %rsp
    movq    -24(%rbp), %r8
    movq    %r8, -48(%rbp)
    call    add
    subq    $8, %rsp
    movq    %rax, -56(%rbp)
    movq    -56(%rbp), %rsi
    leaq    .LC0(%rip), %rdi
    movq    $0, %rax
    addq    $8, %rsp
    call    printf@PLT
    leave
    ret
```

Print functions

```
print:
    pushq   %rbp
    mov     %rsp, %rbp
    testq   $15, %rsp
    jz      is_print_aligned
    pushq $0
    leaq    .LC0(%rip), %rdi
    xor     %rax, %rax
    call    printf
    addq    $8, %rsp
    leave
    ret
is_print_aligned:
    lea     .LC0(%rip), %rdi
    xor     %rax, %rax
    call    printf
    leave
    ret
printstr:
    pushq   %rbp
    mov     %rsp, %rbp
    testq   $15, %rsp
    jz      is_print_alignedstr
    pushq   $0
    leaq    .LC1(%rip), %rdi
    xor     %rax, %rax
    call    printf
    addq    $8, %rsp
    leave
    ret
is_print_alignedstr:
    lea     .LC1(%rip), %rdi
    xor     %rax, %rax
    call    printf
    leave
    ret
```

Memalloc

```
memalloc:
    pushq   %rbp
    mov %rsp, %rbp
    movq    16(%rbp), %rdi
    call malloc
    leave
    ret
```

Object Handling and Method Calls

Python Code

```python
class A:

  def __init__(self):
    self.x: int = 1


def main():
  a: A = A()
  print(a.x)
```

x86 Code

```
        .section        .rodata
    .globl  main
.LC0:
    .string "%d\n"
.LC1:
    .string "%s\n"
    .text
    .globl  __init__
    .type   __init__, @function
A.__init__:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movq    16(%rbp), %r12
    movq    %r12, -8(%rbp)
    subq    $8, %rsp
    movq    $1, -16(%rbp)
    movq    -8(%rbp), %r9
    movq    -16(%rbp), %r8
    movq    %r8, 16(%r9)
    movq    -8(%rbp), %rax
    leave
    ret

    .globl  main
    .type   main, @function
main:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    subq    $8, %rsp
    movq    $24, -16(%rbp)
    call    memalloc
    movq    %rax, -8(%rbp)
    movq    -8(%rbp), %r9
    movq    $1, %r8
    movq    %r8, 8(%r9)
    subq    $8, %rsp
    movq    -8(%rbp), %r8
    movq    %r8, -24(%rbp)
    call    A.__init__
    subq    $8, %rsp
    movq    %rax, -32(%rbp)
    subq    $8, %rsp
    movq    -32(%rbp), %r8
    movq    16(%r8), %r9
    movq    %r9, -40(%rbp)
    movq    -40(%rbp), %rsi
    movq    $0, %rax
    call    print
    leave
    ret
```

5 Error Handling

- If there are spaces after '\' in explicit line joining, lexer will throw an error
- For erroneous strings when there are "' in a multiline string that starts/ends with triple single quotes(same for double quotes), lexer will throw an error
- Improper indentation is reported with the line number where it occurs
- For any redeclaration of a variable in the same scope, the code will throw an error and exit the code
- If there is any N-D list in the program for $N > 1$, then an error will be thrown as only 1-D lists are allowed
- If any array's index is other than *int*, "Iterator has to be of type int" error will be thrown here. No typecasting from $float \to int$ and $bool \to int$ is allowed
- For constant index, it will be checked if the index is not out of bounds. That is, you can only access the index 0 to (size of list - 1); otherwise, the index will be out of bounds
- In our implementation, type casting is only allowed for $int \longleftrightarrow float$, $int \longleftrightarrow bool$ and $bool \longleftrightarrow float$ and will throw an error for all other cases
- Similar type casting is also done for all the elements of the list, as well as function calls and object constructions
- *self* is the only argument allowed with any type hints. For any other argument without a type hint, it will throw an error
- For every function in class, if the first argument is not *self*, it will throw an error
- While creating an object, it will check if the class doesn't have any "*__init__*" function. It will throw an error stating, "Class has no constructor."
- While handling any function call or object construction, if the number of arguments and the type of each argument doesn't match the function definition, an error will be thrown
- If more than one item is passed in the print function, it will throw an error
- If the item passed in the print function is not a variable or a string, it will throw an error
- Similarly, for len, only one argument can be passed, and it has to be a 1-D list
- For range function, up to three arguments are allowed, all of which have to be of type integers without any type casting