Milestone 2

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

1.1 Setup

1. Extract or clone the main folder where all the source files are inside 'milestone1' directory.
2. Download the following libraries in milestone1 directory: graphviz, bison, flex.

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

1.2 Running the shell script

1. Run the wrapper shell script 'main_script.sh' in **milestone1** 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
./src/main_script.sh --input tests/test1.py --verbose
```

2. The wrapper script supports the functions for input, output, help, verbose whenever required.
3. Execute the following command to get an idea of how to run the script:

```
./src/main_script.sh --help
```

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)).
- We assume that the index inside the array will be of type int.
- Range in for loop can include upto 3 arguments, where they are single values or variable but not an expression.

3 Symbol Table

- We have used 2 classes i.e. **SymbolInfo** and **SymbolTable** for the formation of the symbol table.
- SymbolInfo contains the attributes of a particular entry(line number of the declaration, type, size, offset, total number of arguments(in case of a function) and argument number(if it is an argument of a function)) while, SymbolTable contains all the information about that scope which includes the level, scope name, parent pointer which helps in traversing up to the global scope and the map consisting of the SymbolInfos of that scope.
- The global symbol table has the level 0 along with parent NULL.
- We have created levels to signify the different scopes.
- We have also added *"len"* in the global symbol table in "main()", as this is a function with return type "int" that can be called at any time in the code.

4 Type Checking

- **List homogeneity handled:** All the elements within a list should be of the desired type, otherwise the compiler throws a type error.
- Arguments of the functions should be of the allotted type, otherwise the compiler throws an error. Although, it allows the said type casting, such as
- Assignment to the variables should be restricted to the statically typed type.
- We have supported the basic class operations mentioned in the doc and checked the arguments while using a constructor.

5 3AC Code Generation

5.1 Expressions

For expressions like

```
a:int = 4 + 3 / 2 - 7
```

our 3AC code is

```
t0=4
t1=3
t2=2
t3=t1/t2
t4=t0+t3
t5=7
t6=t4-t5
a=t6
```

5.2 for loops

For expressions like

```
i: int = 0
for i in range(1, 10, 2):
    print("hello world")
```

our 3AC code is

```
t0 = 0
i = t0
t1 = 1
t2 = 10
t3 = 2
i = t1
goto L1
L0:
t1 = i + t3
i = t1
L1:
ifz t1 < t2 goto L2
t4 = "hello world"
pushparam t4
stackpointer +xxx
call print 1
stackpointer -yyy
goto L0
L2:
```

Here, if range contains 3 arguments, they are - start parameter(s), end parameter(e), step(g) size i.e. the loop will run (s - e) / g times.
If range contains 2 parameters, they are - start parameter(s), end parameter(e). By default, the step size is 1.

If range contains only 1 parameter, it is the end parameter(e), default step size is 1 and default start parameter is 0.

L0 is the top label which consists the increment operation of for loop.
L1 is the mid label which consists of loop body.
L2 is the end label which is the end of the loop.

5.3 List declaration

For expressions like

```
data: list[int] = [1, 2, 3, 4]
```

our 3AC code is

```
t0 = 1
t1 = 2
t2 = 3
t3 = 4
*(t4 + 0) = t0
*(t4 + 4) = t1
*(t4 + 8) = t2
*(t4 + 12) = t3
data = t4
```

5.4 while loops

For expressions like

```
e:int = 6
while e < 10:
  e+=1
```

our 3AC code is

```
t0 = 6
e = t0
L0:
t1 = 10
t2 = e < t1
ifz t2 goto L1
t3 = 1
t4 = e+t3
e = t4
goto L0
L1:
```

5.5 if - elif - else statements

In the codes below, we say that 'ifz' means if the statement evaluates to zero or False For expressions like

```
i:int = 6
c:int = 0
if i < 2:
  c = 1
  a : int = 50
  if a > 30:
    print(a)
  elif a > 40:
    a = 5
  else:
    i = 1 + a - a
elif i < 4:
  c = 2
elif i < 6:
  c = 3
else:
  c = 4
```

our 3AC code is

```
t0 = 6
i = t0
t1 = 0
c = t1
t2 = 2
t3 = i < t2
ifz t3 goto L1
t4 = 1
c = t4
t5 = 50
a = t5
t6 = 30
t7 = a > t6
ifz t7 goto L3
goto L2
L3:
t8 = 40
t9 = a > t8
ifz t9 goto L4
t10 = 5
a = t10
goto L2
L4:
t11 = 1
t12 = t11+a
t13 = t12-a
i = t13
L2:
```

```
goto L0
L1:
t14 = 4
t15 = i < t14
ifz t15 goto L6
t16 = 2
c = t16
goto L0
L6:
t17 = 6
t18 = i < t17
ifz t18 goto L7
t19 = 3
c = t19
goto L0
L7:
t20 = 4
c = t20
L0:
```

5.6 Array referencing

For expressions like

```
data: list[int] = [1, 2, 3]
x:int = data[1]
```

our 3AC code is

```
t0 = 1
t1 = 2
t2 = 3
*(t3 + 0) = t0
*(t3 + 4) = t1
*(t3 + 8) = t2
data = t3
t4 = 1
t5 = 1 * 4
t6 = data + t5
x = *(t6)
```

5.7 Function Definition

For expressions like

```
def add(a : int, b : int) -> int:
  c : int = 6
  return a + b + c
```

our 3AC code is

```
function add :
    begin func:
    popparam a
    popparam b
    t0 = 6
    c = t0
    t1 = a + b
    t2 = t1 + c
    push t2
    return
    end func:
```

5.8 break and continue

We have handled these conditionals as well