Detailed Tracing, Solutions to Sample Problems

Detailed tracing allows us to explore what happens in memory when a program is executed. We'll simulate what really happens with a fictitious machine-- the numbers in the description below are somewhat arbitrary-- but one that works like the real ones. Here is the info:

1. The translator (e.g., Python) keeps track of a **symbol table** which maps variables to the address where they are stored.

2. Addresses are assigned to each byte (a byte is a set of 8 bits). So address 1000 is for byte 1000 in memory. Integers and float values require 4 bytes of data. Characters require 2 bytes of data. Strings require 2 bytes for each character and 2 bytes for the end of string character (/0). Addresses require 4 bytes of data.

3. Memory has a stack and a heap. The stack is for the variables in the main and in each function. The heap is for dynamic data including string data and list data.

4. The stack begins at address 1000 and the heap at 5000 (note that these are just numbers for our fictitious machine).

5. Start tracing with the main program. When a new variable is defined, add it to the symbol table and store it at the next available address on the stack. If it is a string or list, store the data on the heap, and the address of the data as the value of the variable.

6. When a function is called, a new "frame" is created on the stack, and that function has its own symbol table. Allocate space for each parameter and each local variable. List parameters are sent by reference, scalars by value. Then execute the statements in the function. When the function ends, de-allocate the entire frame on the stack.

For the following program:

```
x = 3
y = 4
z = "abc"
```

This is the trace:

SYMBOL TABLE

SYMBOL	ADDRESS
x	1000
y	1004
z	1008

MEMORY

1000:	3
1004:	4
1008:	5000
5000:	‘a’
5002:	‘b’
5004:	‘c’
5006:	‘\0’

For the following program:

```
x = 3
list = [4,5,6]
z ="d"
```

This is the trace:

SYMBOL TABLE

SYMBOL	ADDRESS
x	1000
list	1004
z	1008

MEMORY

1000:	3
1004:	5000
1008:	5012
5000:	4
5004:	5
5008:	6
5012:	‘d’
5014	‘\0’

For the following program:

```
def totalList(aList,n):
   total = 0
   if n>len(aList):
      n=len(aList)
   for i in range (n):
      total=total+aList[i]
   return total
# main
n=2
list = [3,2,5]
t = totalList(list,n)
```

This is the trace:

SYMBOL TABLE

MAIN

SYMBOL	ADDRESS
n	1000
list	1004
t	1008

totalList

aList	1012
n	1016
total	1020
i	1024

MEMORY

1000:	2
1004:	5000
1008:	5
1012:	5000
1016:	2
1020:	~~0~~ ~~3~~ 5
1024:	~~0~~ ~~1~~ 2
...	
5000:	3
5004:	2
5008:	5
5012:	
5014	