- Authors:
  - Cameron Dunn, dunnc11@mcmaster.ca
  - Rafeed Iqbal, iqbalr8@mcmaster.ca
- Group ID on Avenue: 23
- Gitlab URL: https://gitlab.cas.mcmaster.ca/dunnc11/l-3-assembly-group-23

F2

Memory Allocation:

The visitor GlobalVariables now uses a dict to hold results instead of a set. This allows for the value of variables to be held as well. In turn, the generator StaticMemoryAllocation now uses the values held in the dict to write more efficient memory allocation than before (.Word 'value' is more efficient than using .Block 2)

Constants

Constants are defined in assembly using Equate n and have variable names starting with an underscore and having all capital letters after. This is implemented in the code by a checking the label names during generation in SymbolTable. The name is checked using Python Regex, looking for names which follow the naming scheme for constants. The visitor TopLevelProgram generates instructions for the these constants by checking for a leading underscore, and assigning it a value if it has not already been assigned. Memory is allocated by the generator StaticMemoryAllocation, which once again looks for a leading underscore and then allocates memory accordingly.

Symbol Table:

Pep9 only allows for symbols of 8 characters or fewer. Direct translations of longer variable names or labels is not possible, thus we must use a symbol table. This is implemented in the code by the use of the inbuilt hash() function in python to hash all variable and label names. The hashcode is converted to Hexadecimal and truncated to 7 characters (the 1st character is an x for a variable and an _ for constants). This is used to convert all variable names into 8 character hexcodes in TopLevelProgram when visiting. Since instructions are created by the visitor, the generator EntryPoint does not need to be changed.

Large inputs in fibonacci and factorial

Large inputs cause overflows to occur in Pep9 as the data allocated is not enough to store the large integers being generated

How are overflows handled in programming languages?

Overflow errors in programming languages are typically detected by the programming language's runtime environment through error checking mechanisms. When an overflow error occurs, the runtime environment will typically raise an exception and halt the program's execution. In some cases, the programmer can specify how the runtime environment should handle an overflow error by providing a handler for the exception. For example, in Python, the programmer can use a try/except block to handle an overflow error.