CS420: Compiler Design

Fall, 2019

Optional Feature 1 : Memory Management

- **Memory Management**

  Students who choose this option should implement memory management feature in the program. Each dynamically allocated memory space must be resident in 1kB linear heap memory space. Program also must be able to handle memory fragmentation issue.

  1. **Static allocation**

     The interpreter must count the number of variables and calculate memory usage of variables. These must be printed on console at the start of the interpretation.

  2. **Dynamic allocation**

     - **Memory structure**: The interpreter must construct and manage a memory structure as heap spaces. The characteristics of memory structure is below.
       - 1kB linear memory capacity
       - Every allocation must have a continuous space starting with a distinct address. This is so called as a chunk. Chunks must not be separated into several discontinuous regions.
       - Each allocated chunk must contain the real data of the correspondent variable. The chunks must be managed in a certain policy with a structure. Implementation design on the structure is up to you.

◆ **Actions**

- Allocation: must be performed on *void* malloc(int size_t)* function. A *size_t*-sized chunk must be generated in the memory space, and the correspondent address must be returned.
- Deallocation: must be performed on *void free(void* ptr)* function. The correspondent chunk of the address ptr must be released from the memory space.
- Defragmentation: must be performed on memory fragmentations.

◆ **Command**

- **mem** command: This command must count and print the number of allocated chunks and memory usage.

- **Expected Result for the interpreter implementation**

  The implemented feature is expected to operate as an example below

Example input code	Interpreter input commands and results

1 `int main(void)`
2 {
3 `  char *address1, *address2, *address3, *address4, *address5;`
4
5 `  address1 = malloc(111);`
6 `  address2 = malloc(222);`
7 `  address3 = malloc(333);`
8 `  free(address2);`
9 `  address4 = malloc(444);`
10 `  address5 = malloc(555);`
}
(In this example, the interpreter starts at the top of main function, line 2)
 	
Static allocation : 4, 16
>> **mem**
Dynamic allocation : 0, 0
>> **next** 5
>> **mem**
Dynamic allocation : 2, 333
>> **next**
>> **mem**
Dynamic allocation : 3, 666
>> **next**
>> **mem**
Dynamic allocation : 2, 444
>> **next**
Defragmentation operated
>> **mem**
Dynamic allocation : 3, 888
>> **next**
Out of memory
>> **next**
End of program
>>

Requirement Specification

Non-functional	
Defragmentation	When a newly being allocated memory space cannot reside in the heap space because of memory fragmentation, the allocated memory spaces must be defragmented in a certain policy. At the same time, the interpreter console must print "*Defragmentation operated*".
Memory overflow handling	When the size of a newly being allocated memory space is larger than the sum of the all remaining spaces, the interpreter console must print "*Out of memory*" and stop interpretation.
Functional	
Memory management	The interpreter must manage allocate and deallocate chunks in
malloc function	The interpreter must try to allocate a *size_t*-sized chunk on *void* malloc(int size_t)* function. The correspondent address on the memory space must be returned. When *size_t* < 1, print "*size_t should be a positive integer*".
free function	The interpreter must try to deallocate on *void free(void* ptr)* function. When the address ptr is not allocated, print "*not allocated address*".
mem command	Interpreter should accept the command below via CLI. This command must print "*Dynamic allocation :* [a], [b]". [a] : the number of allocated chunks [b] : the total size of allocated chunks