Test 2 practice paper B (theory)

This assessment contains questions that allow partial credit. Dismiss

11 OF 11 QUESTIONS REMAINING

Test Content

Question 1

3 Points

Which of the following memory segments in Hack VM would be most appropriate, when compiling from a high-level language such as C, to store the fields of a global variable?

- A. this
- B. that
- C. pointer
- D. local
- E. static
- F. it depends on context

Question 2

5 Points

In an **assembler**, which of the following tasks would typically be carried out during lexing?

a) Converting the program from a string into a sequence of tokens.

b) Constructing a symbol table mapping labels to memory addresses.

c) Constructing a symbol table mapping variables to memory addresses.

d) Converting each line of assembly into machine code.

e) Optimising the resulting machine code.

Question 3

4 Points

What are the binary values of the *comp*, *dest*, and *jump* operands for the C-instruction AM=!M? Your answers should include any leading 0s, where appropriate.

comp = Enter answer , dest = Enter answer , jump =

Enter answer

Question 4

5 Points

In the context of memory allocation, which of the following are possible consequences of fragmentation? If relevant, you may assume the memory allocation algorithm is the version covered in lectures with coalescence of free segments but without bins ("attempt 3").

a) Existing data in memory becomes corrupted and is no longer accessible.

b) Allocating a large new memory segment fails, even though there is enough usable space for it in memory.

c) Allocating a two-word memory segment fails, even though half of memory is free.

d) Allocating a new memory segment takes much longer than normal.

e) Freeing a memory segment takes much longer than normal.

Question 5

4 Points

What **signed decimal** value will the following sequence of VM operations leave on top of the stack?

```
push constant 4
push constant 5
sub
not
push constant 0
eq
```

Add your answer

Integer, decimal or E notation allowed

Question 6

5 Points

Complete the following sentences accurately.

a) Instructions are more likely to be of variable length in a [ ] architecture.

b) Compared to ARM, MIPS is more of a [ ] ISA and x86-64 is more of a [ ] ISA.

c) If the same C code is compiled to machine code in a CISC ISA and a RISC ISA, then one would expect the [ ] machine code to have more instructions.

d) [ ] ISAs typically use memory less efficiently.

Question 7

4 Points

Which of the following fragments of Hack assembly code could contain a data hazard for the purpose of pipelining?

- **A.**
  ```
  M=!M
  A=!A
  D=!D
  ```
- **B.**
  ```
  D=M+1
  A=M+1
  ```
- **C.**
  ```
  M=A-D
  A=A+D
  D=0
  ```
- **D.**
  ```
  A=A+1
  M=M+1
  D=D+A
  ```
- **E.** None of the other options.
- **F.** More than one of the other options.

Question 8

5 Points

Below is part of the Jack grammar:

⟨expression⟩ ::= ⟨term⟩, {('+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '='), ⟨term⟩};

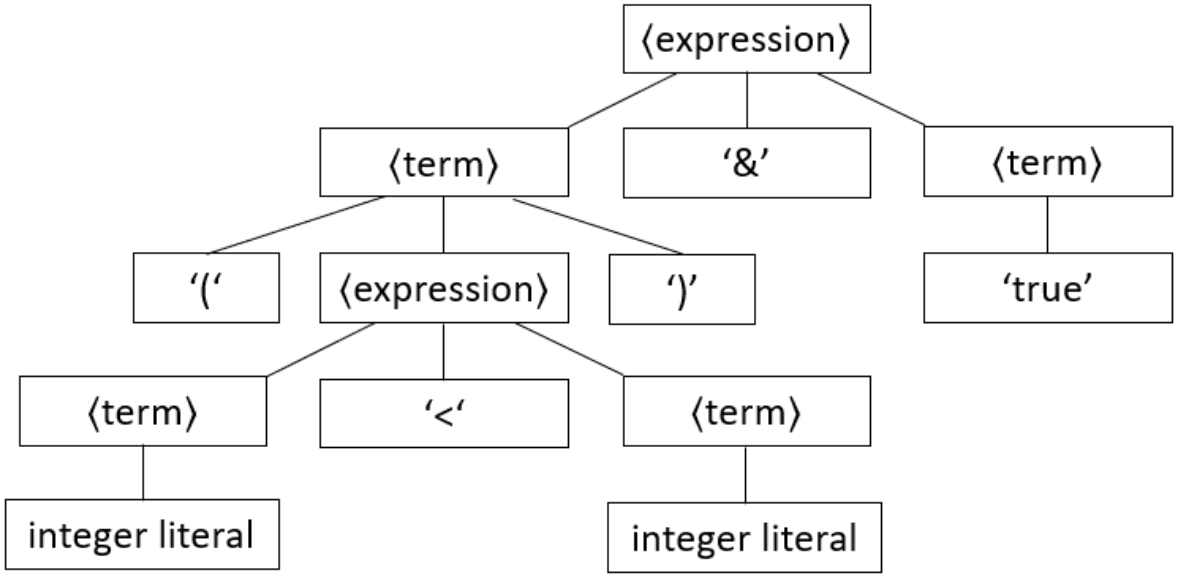
⟨term⟩ ::= integer literal | string literal | 'true' | 'false' | 'null' | 'this' | identifier, ['[', ⟨expression⟩, ']'] | '(', ⟨expression⟩, ')' | (('-' | '~'), ⟨term⟩) | ⟨subroutineCall⟩;

⟨subroutineCall⟩ ::= identifier, ['.', identifier], '(', ⟨expressionList⟩, ')';

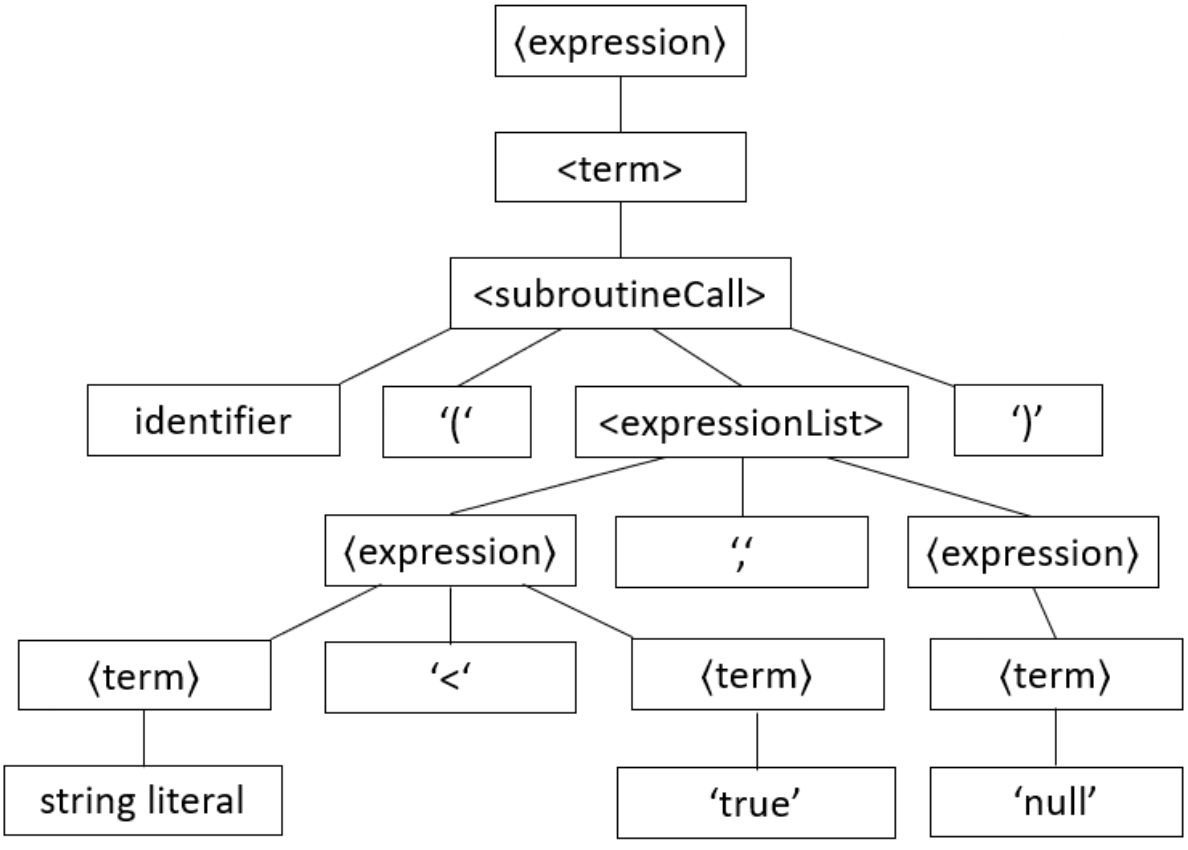
⟨expressionList⟩ ::= [⟨expression⟩, {',', ⟨expression⟩}];

Which of the below is **not** a valid CST for a valid <expression>?

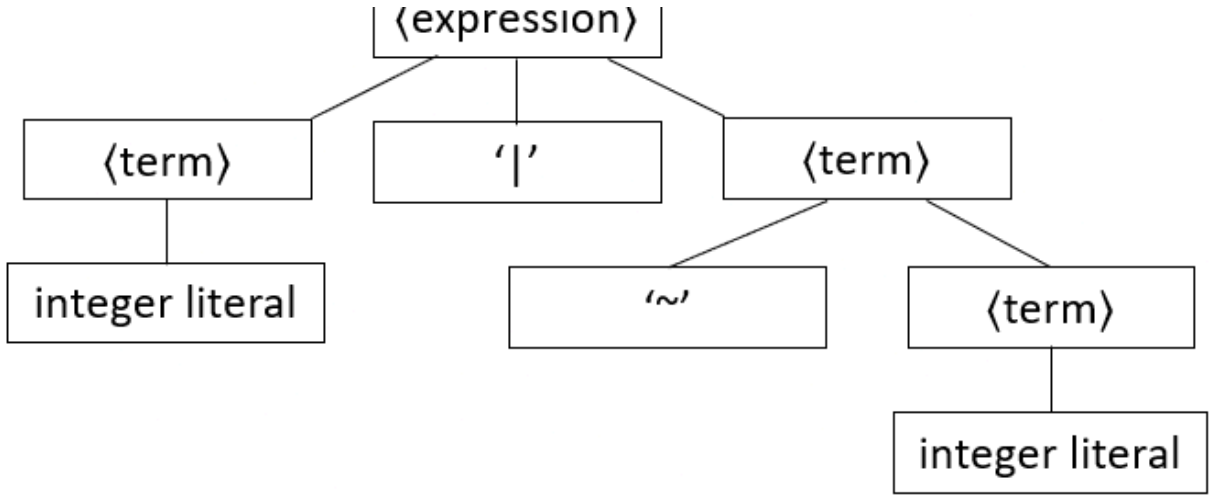
A.



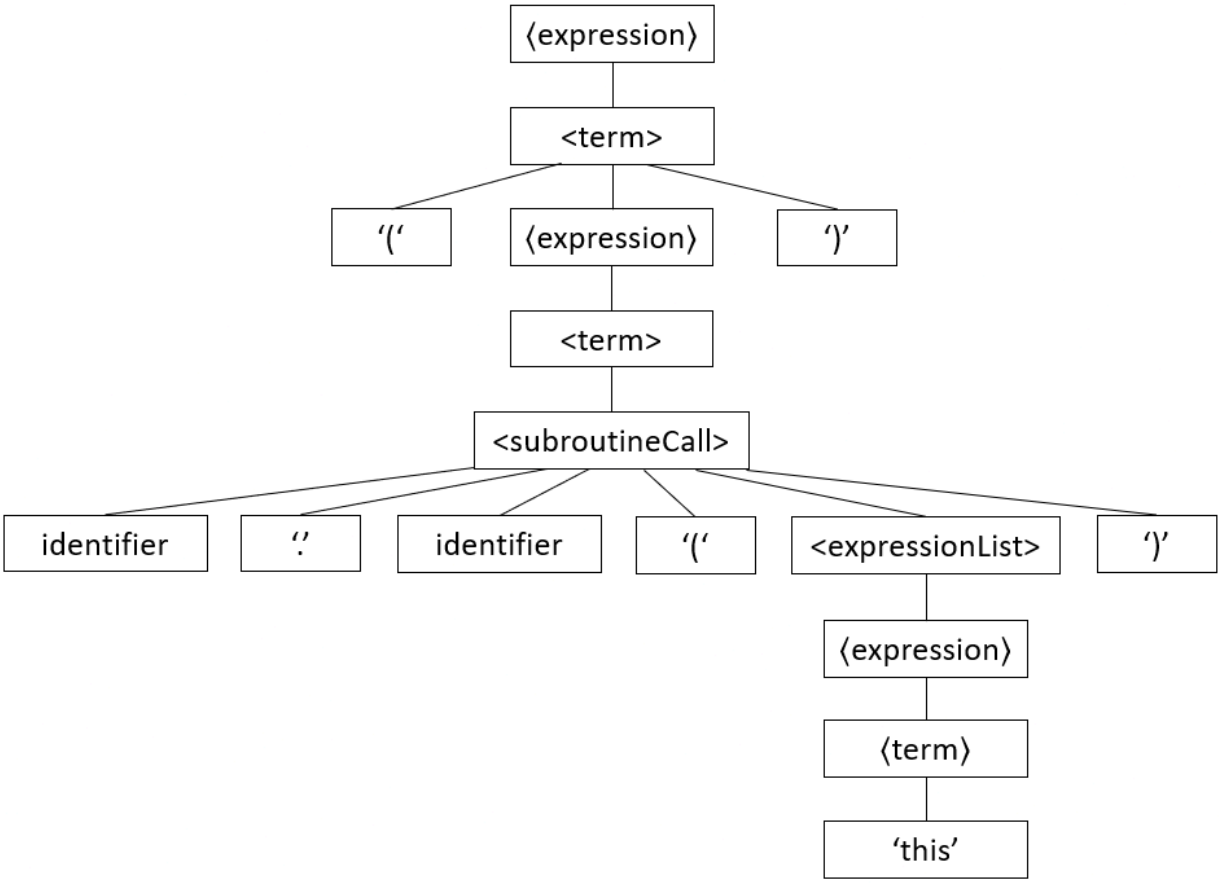
B.



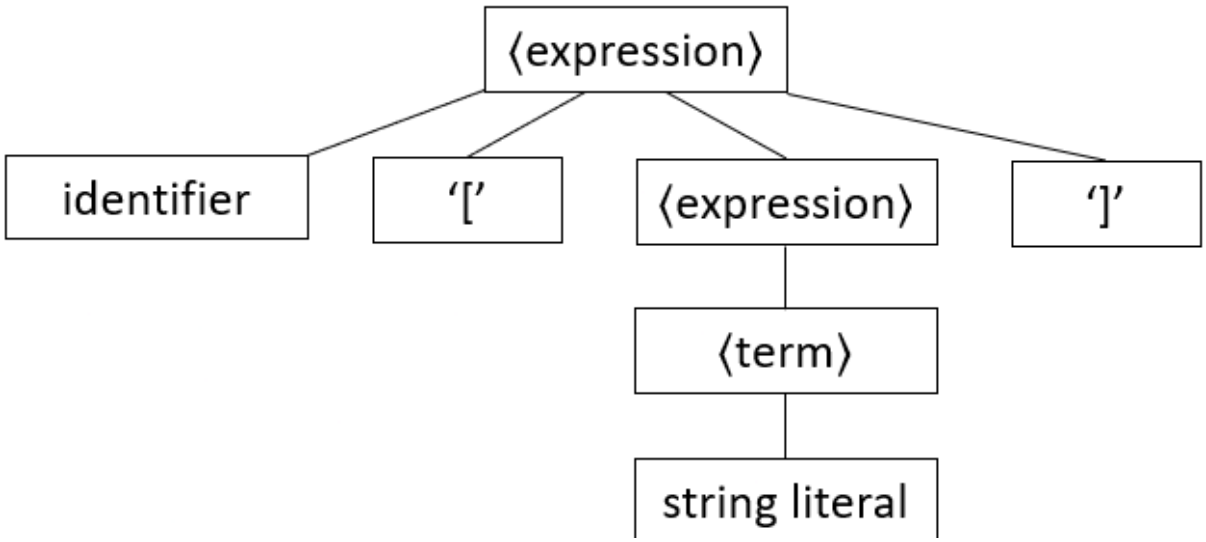
C.


⟨expression⟩
⟨term⟩
'|'
⟨term⟩
integer literal
'~'
⟨term⟩
integer literal

D.


⟨expression⟩
<term>
'('
⟨expression⟩
')'
<term>
<subroutineCall>
identifier
'.'
identifier
'('
<expressionList>
')'
⟨expression⟩
⟨term⟩
'this'

E.


⟨expression⟩
identifier
'['
⟨expression⟩
']'
⟨term⟩
string literal

F. More than one of the other options.

G. None of the other options.

Question 9

5 Points

You are reading the documentation for a new programming language called Crust. The documentation tells you that "In Crust, all strings are stored as pointers to heap memory". From this, what can you tell about the following statements about strings in Crust?

a) If you define a string variable inside a function, and then return that string, your code will break.

[ ]

b) If you define a string variable, it will use the ASCII character set. [ ]

c) If you define a string variable and pass it to a function, and that function modifies the string, that modification will persist after the function returns. [ ]

d) If you define two string variables myString and myOtherString, then set myOtherString = myString, then myOtherString and myString will be independent copies of each other. You can modify myString without modifying myOtherString, and vice versa. [ ]

e) If you define a string variable, then at some point the memory associated with it will need to be freed with a call to something like C's free function or Jack's Memory.deAlloc function. This call might be manual or automatic, depending on how Crust works. [ ]

Question 10

2 Points

Consider the following proposed grammar for baby noises, given below in EBNF form with tokens 'A' and 'B'.

⟨a⟩ ::= ⟨b⟩, 'A' | [⟨a⟩], 'A';

⟨b⟩ ::= ⟨b⟩, ⟨b⟩, ⟨b⟩, {⟨b⟩} | ⟨a⟩, ⟨a⟩ | 'B';

Which of the following strings is **not** a valid ⟨a⟩ under this grammar?

- **A.** A
- **B.** BBBBBAAAAAAA
- **C.** AABBA
- **D.** BABABABABABAA
- **E.** BAABAABAABAABAAA
- **F.** None of the other options.
- **G.** More than one of the other options.

Question 11

8 Points

You are trying to write Hack VM code that will dump each of the user's keypresses into RAM [0x1000]--RAM[0x1FFF]. The first keypress should be entered into RAM[0x1000], the second into RAM[0x1001], and so on. On entering a keypress into RAM[0x1FFF], you want your code to wrap around and start entering keypresses into RAM[0x1000] again. You also want your code to register each distinct keypress only once - if the user holds a key down, it should be recorded in memory once only.

Fill in the blanks in the following Hack VM program to accomplish this goal. Each Hack VM instruction contains 1-3 words. If you want to fill a line with an instruction containing only one word (like "add"), then choose "add", "N/A", and "N/A" from the drop-downs in that order. Likewise for instructions containing only two words.

// initialise values
push constant 0
pop local 0
push constant 4096
pop pointer 0
[ ] [ ] [ ]
pop pointer 1
[ ] [ ] [ ]
// test if the keypress is new
push that 0
push local 0
eq
if-goto main_loop_start
// store keypress in local 0 and next location in RAM[0x1000]--RAM[0x1FFF]
push that 0
pop local 0
push local 0
[ ] [ ] [ ]
// increment location to store next new keypress
[ ] [ ] [ ]
push constant 1
add
pop pointer 0
// check if 0x1FFF has been reached
push constant 8192
push pointer 0
eq
[ ] [ ] [ ]
[ ] [ ] [ ]
// wrap around location to store next keypress
push constant 4096
pop pointer 0

```
goto main_loop_start
```

Questions Filter (11) ▼

Save and Close

Submit