CMSC330 Spring 2017 Final Exam Solution

Name (PRINT YOUR NAME as it appears on gradescope):

____

Discussion Time (circle one) **10am 11am 12pm 1pm 2pm 3pm**

Discussion TA (circle one) **Aaron Alex Austin Ayman Daniel Eric**

Greg Jake JT Sam Tal BT Vitung

Instructions

- The exam has 19 pages (front and back); make sure you have them all.
- Do not start this test until you are told to do so!
- You have 120 minutes to take this exam.
- This exam has a total of 130 points, so allocate 55 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

#	**Problem**	**Score**
1	PL Concepts	/10
2	Lambda Calculus	/8
3	OCaml	/30
4	Ruby	/14
5	Prolog	/22
6	Regexps, FAs, CFGs	/22
7	Parsing	/8
8	Operational Semantics	/6
9	Security	/10
	TOTAL	/130

(This page intentionally left blank)

1. Programming Language Concepts (10 points)

A. (6 points) Circle True or False for each statement.

T / F	Ruby uses implicit declarations
T / **F**	Ruby code blocks are equivalent (have the same power as) OCaml closures
T / F	Backtracking is necessary for Prolog programs to provide multiple answers
T / **F**	A function call in OCaml can overflow the stack if it uses tail recursion
T / **F**	Multiple implementations of an abstract data type (ADT) can safely interoperate
T / F	ADTs are only possible in languages with static typing

B. (2 points) Predicate `max` takes 3 integer arguments and succeeds if the third argument is the maximum of the first two.

```
max(X,Y,Z)  :-  X  =<  Y, !, Y  =  Z.
max(X,Y,X).
```

The cut in above code is a (circle the right answer) Green cut **Red cut**

C. (1 point) Which memory management technique may have difficulty freeing cyclic structures?

a) Mark & Sweep Garbage Collection
b) Stop & Copy Garbage Collection
c) Reference Counting
d) Malloc & Free

D. (1 point) A type error in your code has gone unnoticed during testing because the line containing the error was never executed. The language most likely uses (circle one)

Static Typing **Dynamic Typing**

2. Lambda Calculus (8 pts)

A. (1 pt) The lambda expression `λf.λy.f y p` is alpha-equivalent to which one of the following lambda expressions (circle the correct one)?

```
a) λg.λy.g (y p)
b) λf.(λy.f (y p))
c) (λf.λy.f y) p
```
`d) (λf.λx.(f x) p)`

B. (1 pt) Fully reduce the following lambda calculus term

`(λa.λc.c) z -->` **`λc.c`**

C. (3 pts) Fully reduce the following lambda calculus term

`((λa.λc.λb.a c b) c) z`

λb. c z b

D. (3 pts) In the standard encoding, `true` is encoded as `λx.λy.x`, while `false` is encoded as `λx.λy.y`, and `if a then b else c` is encoded as `a b c`. So, for example, `if true then true else false` is encoded as `(λx.λy.x) (λx.λy.x) (λx.λy.y).`

Give the lambda calculus encoding of the `not` function such that `not true` reduces to `false`; i.e., `not (λx.λy.x)` would reduce to `(λx.λy.y)` (and `not false` reduces to `true`).

λa. a false true

Or, more specifically: λa. a (λx.λy.y) (λx.λy.x)

Equivalent alternative: (λa.λb.λc. a c b)

3. OCaml (30 pts)

A. (8 pts) Consider the following function

```
let rec f l =
  match l with
      [x] -> x
    | h::t ->
      let m = f t in
      if h > m then h else m
```

1) (2 pts) What is the type of `f`?[1]

‘a list -> ’a

The next three questions ask you to consider the result of execution. For these, and for others below, if execution proceeds normally, indicate the final result. If it raises an exception or it loops infinitely, indicate that. If there is a type error, indicate that.

2) (2 pts) What is the result of executing `f [1;2;"3"]` ?

Type error

3) (2 pts) What is the result of executing `f [3;2;7]` ?

7

4) (2 pts) What is the result of executing `f []` ?

Match_failure exception

[1] Hint: the comparison function > is polymorphic.

B. (7 pts) The following type definitions encode a finite automaton.[2] They resemble the definitions from your project 4, but use OCaml records, rather than tuples.

```
type state = int
type transition = { curr: state; sym: char; next: state }
type fa = { initial: state; final: state list; delta: transition list }
```

Here are two example finite automata (i.e., `mach0` and `mach1` both have type `fa`):

```
let mach0 =
  { initial = 0;
    final = [1];
    delta = [
      { curr = 0; sym = 'a'; next = 0 };
      { curr = 0; sym = 'b'; next = 1 }
    ]
  }
```
	```
	let mach1 =
	  { initial = 0;
	    final = [0;1];
	    delta = [
	      { curr = 0; sym = 'a'; next = 0 };
	      { curr = 0; sym = 'b'; next = 1 };
	      { curr = 1; sym = 'a'; next = 2 };
	      { curr = 1; sym = 'b'; next = 1 };
	      { curr = 2; sym = 'a'; next = 2 };
	      { curr = 2; sym = 'b'; next = 2 }
	    ]
	  }
	```

Consider the following function:

```
let foo m =
  let rec aux ss =
    match ss with
      [] -> false
    | s::ss0 -> s = m.initial || aux ss0 in
  aux m.final
```

1) (2 pts) What is the type of `foo`?

fa -> bool

2) (2 pts) What is the result of evaluating `foo mach0`?

false

[2] These automata are meant to be deterministic, but that actually doesn't matter for this problem. (The `fa` type definition prevents automata from having epsilon transitions, but there's nothing stopping you from making an `fa` transitioning from one state to multiple other states on the same input character.)

3) (2 pts) What is the result of evaluating `foo mach1`?

true

4) (1 pt) What does `foo`'s result say about the strings the automaton will accept?

One of them is the empty string

C. (7 pts) Consider the following function:

```
let bar m =
  let aux x l = if List.mem x l then l else x::l in
  List.fold_left (fun ss t ->
    let ss' = aux t.curr ss in
    let ss'' = aux t.next ss' in
    ss'') (aux m.initial m.final) m.delta
```

Here, the function `List.mem` has type `‘a -> ‘a list -> bool`; executing `List.mem x l` returns `true` if `x` appears anywhere in list `l`, and returns `false` otherwise. `List.fold_left` is the `fold` function we have used in class, having type `(‘a -> ‘b -> ‘a) -> ‘a -> ‘b list -> ‘a`.

1) (2 pts) What is the type of the `bar` function?

fa -> state list

2) (2 pts) What is the result of evaluating `bar mach0`?

[0;1]

3) (2 pts) What is the result of evaluating `bar mach1`?

[2;0;1]

4) (1 pt) What information about the given automaton is `bar` computing?

All of the states of the automaton

D. (8 pts) Write the function `is_dead_state`, having type `fa -> state -> bool`, where `is_dead_state m s` returns `true` if `s` is a dead state in `m` (and `false` otherwise). A dead state is one in which all transitions from the state go back to itself, and the state is not a final state. As such `is_dead_state mach1 2 = false` but `is_dead_state mach1 3 = true`.

```
let is_dead m s =
      if List.mem s m.final then false
      else
            let rec aux trans =
                  match trans with
                  []->true
                  |h::t->if s = h.curr && s <> h.next then false
                    else aux t
in aux m.delta
```

Or (there are many more)

```
let is_dead m s =
  if List.mem s m.final then false
  else
    List.fold_left
      (fun isdead t -> isdead && (if t.curr = s then t.next = s else true))
      true m.delta
```

For your reference, here are the type definitions again:

```
type state = int
type transition = { curr: state; sym: char; next: state }
type fa = { initial: state; final: state list; delta: transition list }
```

4. Ruby (14 pts)

We want to transfer your midterm grades from gradescope to grades server. We have a course roster file `roster.csv`. The following code reads in the file, creating a `Student` object for each non-comment line it sees, and storing that object in the `students` hash.

```
class Student
  attr_accessor :last, :first, :uid, :dirid, :score
end
students = {}
File.open("roster.csv", "r") do |f|
  f.each_line { |line|
    rec = getdata(line)  # you will implement this
    if not rec.empty? then
      s = Student.new
      s.last = rec[0]
      s.first = rec[1]
      s.uid = rec[2]
      s.dirid = rec[3]
      s.score = 0
      students[s.dirid] = s
    end
  }
end
```

The `roster.csv` file is formatted as comma-separated-values (CSV) as shown here:

```
#Lastname,Firstname,UID,DirectoryID
SquarePants,SpongeBob,123456789,spants
Cheeks,Sandy,114123398,scheeks
Laguna,Jack Kahuna,123456778,jlag34
…
```

We also have a `gradescope.csv` file, which has a different format. It mentions a student's *first and last name*, *directory ID*, and *score*:

```
#Name(first last),DirectoryID,Score
SpongeBob SquarePants,spants,88
Sandy Cheeks,scheeks,93
Jack Laguna,jlag34,77
…
```

A. (4 pts) Implement the `getdata(line)` method called from the code above. It should return an array containing the relevant student values for the given `line`. If it sees a line starting with `#` it should return the empty list. You do not need to worry about malformed input.

```
def getdata(line)

  if line.start_with?("#") then
    []
  else
    line.chomp.split(",")
  end

Alternative (use regular expression):
if /^#/ =~ line then
      []
else
      line.split(‘,’)
end

end
```

B. (4 pts) The following is code that reads in the `gradescope.csv` file (whose format is given above). Fill in the missing code, which should take the read-in `rec` and use it to update the `score` field of the relevant `Student` object in the `students` hash.

```
File.open("gradescope.csv", "r") do |f|
  f.each_line { |line|
    rec = get_gs_data(line) # reads the line from the file; not shown
    if not rec.empty? then
      # rec[0] is firstlast, rec[1] is dirid, rec[2] is score
      # fill in code below to update student objects with a score

      dirid = rec[1]
      if students[dirid] then
        students[dirid].score = rec[2]
      end





    end
  }
```

```
end
```

C. (4 pts) Finally, the grades server requires the data in the following format:

```
#DirID,Score
spants,88
scheeks,93
jlag34,77
```

Use a code block to iterate over the `students` hash to produce output in the above format (you don't have to print the comment `#DirID,Score`).

```
students.each {|k,v|
  printf "%s,%s\n",v.dirid,v.score
}
```

D. (2 pts) Sometimes students enter their gradescope UserId improperly: instead of using their Directory ID, they use their UID. Which of the following choices best describes what will happen when running the code above, if they do this?

a) It will work fine -- the output will be the same in both cases
b) It will work, but instead of outputting the directory ID and the score, it will output the UID and the score
c) It will fail to associate the score with the proper user, and thus output a score of 0
d) It will fail with an exception

Either c or d could be correct, depending on what their code does; a or b definitely are not. (Give credit for either c or d; don't bother looking at their code.)

5. Prolog (22 pts)

A. (14 pts) Consider the following predicates:

```
plays(flagTwirlers, flags).
plays(plankton, keyboard).
plays(squidward, clarinet).
plays(patrick, mayo).
plays(patrick, drums).
plays(spongebob, drums).
plays(spongebob, spatula).
```

```
twoPlayers(X) :- plays(Y, X), plays(Z, X), Y \= Z.
talented(X) :- plays(X, Y), plays(X, Z), Y \= Z.

band1([ ], [ ]).
band1([H | T], [X | Y]) :-
    talented(H), !, plays(H, X), band1(T, Y).

band2([ ], [ ]).
band2([H | T], [X | Y]) :-
    plays(H, X), band2(T, Y), \+ member(H, T).
```

For each query, state whether the query is `true` or list all of the substitution(s) that make the query true. If the query is `false` or if there are no substitutions, write `false`. For full credit (2 pts each), queries with multiple substitutions must have their substitutions listed in the correct order.

a) `?- plays(patrick,X).`

X = mayo;

X= drums.

b) `?- twoPlayers(X).`

X=drums;

X=drums.

c) `?- talented(squidward).`

false.

d) `?- band1([patrick, flagTwirlers], X).`

false.

e) `?- band1(X, [drums, mayo]).`

X=[patrick,patrick]

f) `?- band2(X, [drums, drums]).`

X=[patrick,spongebob];

X=[spongebob,patrick].

g) `?- band2([spongebob, plankton, flagTwirlers], Y).`

Y=[drums,keyboard,flags];

Y=[spatula,keyboard,flags].

B. (6 pts) Write the Prolog predicate `solo(Player,Band)` which is `true` when `Player` appears **exactly once** in `Band`. Backtracking may produce multiple answers. You may use standard library predicates.

```
?- solo(squidward,[squidward,patrick,patrick]).
true.
?- solo(patrick,[squidward,patrick,patrick]).
false.
?- solo(X,[squidward,patrick,flagTwirlers]).
X = squidward;
X = patrick;
X = flagTwirlers.
```

```
count(X,[],0).
count(X,[H|T],N) :- X = H, count(X,T,N2), N is N2+1.
count(X,[H|T],N) :- X \= H, count(X,T,N).
solo(P,B) :- member(P,B),count(P,B,1).
(or)
solo(X,[X|T]) :- \+ member(X,T).
solo(X,[Y|T]) :- solo(X,T), X \= Y.
(or)
solo(P,B) :- member(P,B), select(P,B,NB), \+ member(P,NB).
(or)
solo(P,B) :- append(X,[P|A],B), \+ member(P,X), \+ member(P,A).
```

C. (2 pts) Consider the following two implementations of the list membership predicate `member(X,L)`.

(a)	(b)

`member(X,[X|L]) :- !.`
`member(X,[Y|L]) :- member(X,L).`
	`member(X,[X|L]).`
	`member(X,[Y|L]) :- member(X,L).`

Give a query that will produce a different answer for implementation (a) than for implementation (b).

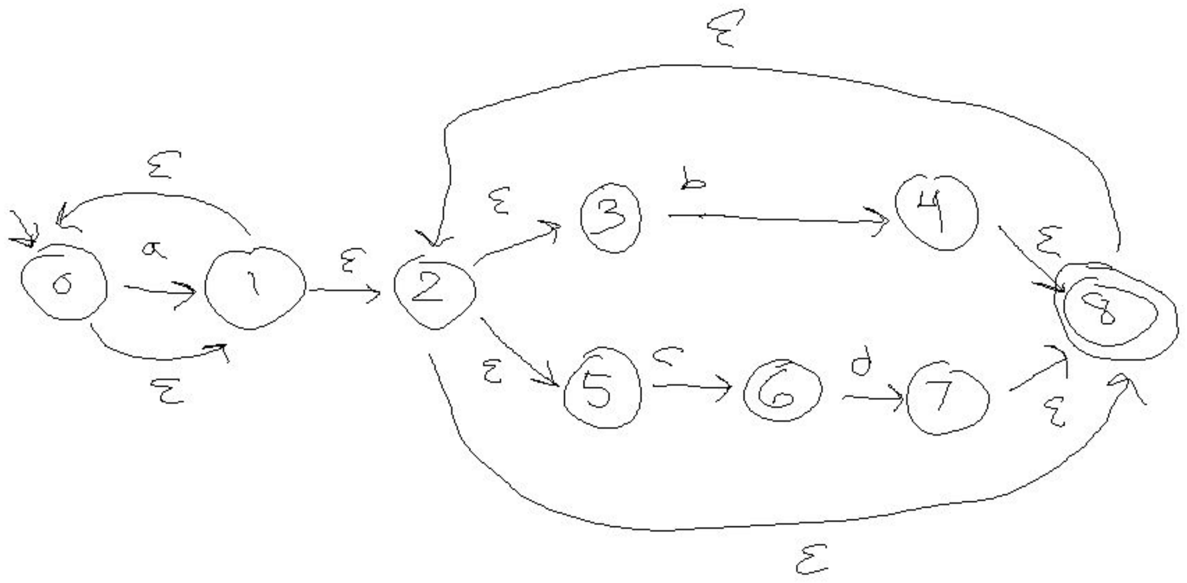
Form of answer is
`member(_,[...])` for any list ... that has more than one element

6. Regexps, FAs, CFGs (22 pts)

A. (2 pts) Give a regular expression for the set of all binary sequences whose first and last bits are different. For example, 1110 has different first and last bits while 1011 does not.

(0(0|1)*1) | (1(0|1)*0)

B. (4 pts) Draw an NFA for the regular expression a*(b|cd)*.



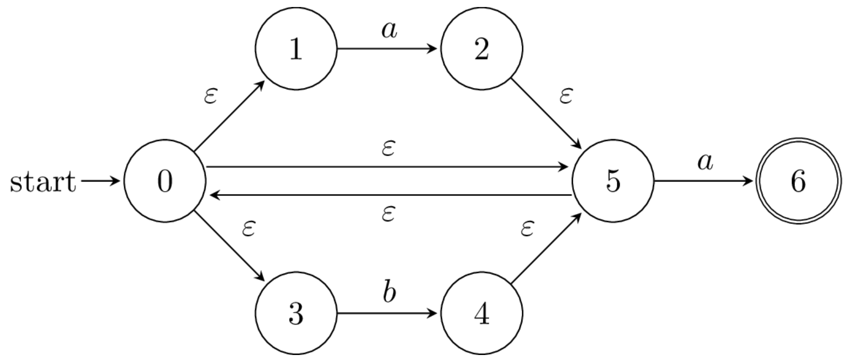
C. (4 pts) Show that the following context-free grammar is ambiguous by giving two leftmost derivations of a string in the language of the grammar.

$$S \rightarrow \mathrm{a}S \mid \mathrm{a}S\mathrm{b}S \mid \mathrm{c}$$

S->aS->aaSbS->aacbS->aacbc

S->aSbS->aaSbS->aacbS->aacbc

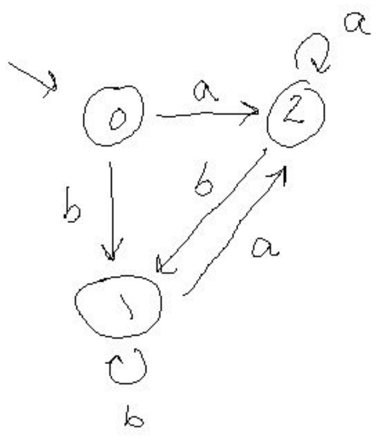
D. (8 pts) Consider the following NFA over the alphabet {a,b}.

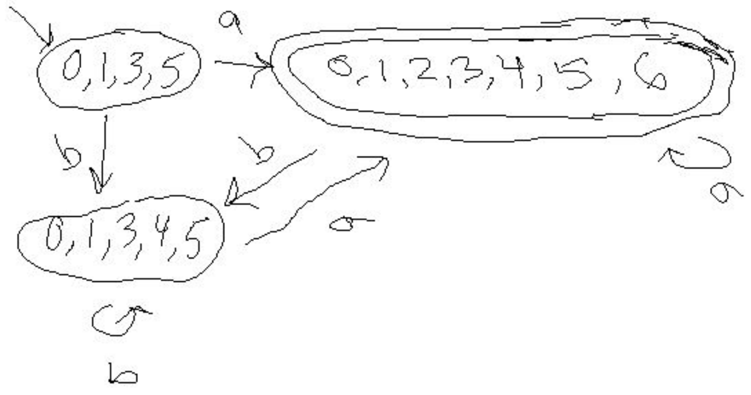


a. (3 pts) Give a regular expression for the set of strings accepted by the NFA.

(a|b)*a

b. (5 pts) Convert the NFA into an equivalent DFA.





E. (4 pts) The following context-free grammar for partial regular expressions over the alphabet {a,b} has left-recursive productions. Rewrite the grammar so that **no productions are left-recursive** and **concatenation has lower precedence than the Kleene star**.

$R \rightarrow RR \mid R^* \mid A$
$A \rightarrow \mathrm{a} \mid \mathrm{b}$

R -> TR | T
T -> AP
P -> *P | epsilon
A -> a | b

7. Parsing (8 pts)

Consider the following code, which implements a recursive descent parser in OCaml in the same style as example code we provided for your project 5.

```
exception ParseError of string
let tok_list = ref [];;

let lookahead () =
  match !tok_list with
      [] -> raise (ParseError "no tokens")
             | (h::t) -> h

let match_tok a =
  match !tok_list with
      (* checks lookahead; advances on match *)
      | (h::t) when a = h -> tok_list := t
      | _ -> raise (ParseError "bad match")

let rec parse_S () =
  let t = lookahead () in
  match t  with
  | 'a' ->
      match_tok 'a';
      parse_S ();
      match_tok 'a'
  | 'b' ->
      match_tok 'b';
      parse_S ();
      match_tok 'b'
  | '+' -> match_tok '+'
  | _ -> raise (ParseError "parse error")
;;
```

A. (4 pts) Write the CFG that expresses the strings accepted by this parser.

S -> aSa | bSb | +

B. (4 pts) Give two valid strings that belong to this language

a+a

b+b

8. Operational Semantics (6 pts)

A. (2 pts) The operational semantics judgment `A; e ⇒ v` states that *"under environment* `A`*, expression* `e` *evaluates to value* `v`*."* Write an operational semantics rule that states *"under environment* `A`*, expression* `if e1 e2 e3` *evaluates to* `v` *if under* `A` *expression* `e1` *evaluates to* `0`*, and under* `A` *expression* `e3` *evaluates to* `v`*."*

```
A; e1 ⇒ 0
A; e3 ⇒ v
----------------
A; if e1 e2 e3 ⇒ v
```

B. (4 pts) Plankton's computer wife Karen has been having some medical issues lately. Being that you're a professional SmallC semantics doctor, he asks you to take a look at her corrupted version of the SmallC operational semantics and see if you can make a diagnosis.

Recall that an environment `A` is a map from variables to integers. The SmallC judgment `A; s ⇒ A'` states that under environment `A`, statement `s` executes and produces a new environment `A'`. (The environment `A'` might differ from `A` in the case that there are assignments in `s`.) The SmallC judgment `A; e ⇒ n` states that under environment `A` expression `e` evaluates to some number `n`.

You figure out that each of the following rules has one mistake. Circle the mistake in each rule and label it with a short (no more than a few words) explanation of what's wrong.

Sequence	Integer Assignment
$\dfrac{\begin{array}{c}A;\ s_1 \Rightarrow A_1 \\ A;\ s_2 \Rightarrow A_2\end{array}}{A;\ s_1;s_2 \Rightarrow A_2}$	$\dfrac{\begin{array}{c}A(x) = n \quad \text{(for some } n\text{)} \\ A;\ e \Rightarrow n'\end{array}}{A;\ x = e \Rightarrow A[x \mapsto e]}$

For the first: A; s2 ⇒ A2 should be A1; s2 ⇒ A2

For the second: the conclusion A; x = e ⇒ A[x ↦ e] should be A; x = e ⇒ A[x ↦ n']

9. Security (10 pts)

Suppose Spongebob constructed a web service to let people know which friends are active on his machine. The web server receives the request as a parameter `user`, which is given to the following Ruby method:

```
def is_active(user)
 user.gsub!(/[;&|!#])   # blacklist: sufficient to check for or sanitize [;&|]
 (or, better:)
 user.gsub!(/\W/,' ')   # whitelist only alpha characters

 found = system ("ps aux | cut -c1-#{user.length} | grep #{user} > /dev/null")

 if found then
   puts "#{user} is active"
 else
   puts "#{user} is inactive"
 end

end
```

The `system()` call invokes a shell command. This command checks the OS process table to see if the given user has an active process by calling `ps` to get all active processes, calling `cut` to truncate the output to just the first *N* columns where *N* is the length of `user`, and calling `grep` to look in this output for the given `user`.

A. (1 pt) SpongeBob is considering the "network user" attack model, and realizes his code might be vulnerable to an attack. What kind of attack is it (circle one)?

Command injection **XSS** SQL injection Buffer overflow

B. (3 pts) Give a value of `user` (i.e., a particular string) that will exploit the vulnerability in `is_active` to compromise the *integrity* of SpongeBob's system.

"foo; echo aha > foo"

Basically: Anything following a semicolon that modifies the system in some way

C. (3 pts) Modify the `is_active` method above with a fix to the vulnerability. Be as precise as possible, but you may write pseudocode for Ruby methods you can't remember.

D. (1 pt) Suppose SpongeBob decides to password-protect his service to reduce the chances of harm. For this purpose he sets up a database with a `Users` table that has a row for each user containing the user's `name` and `password`. Web clients provide this data via a web form which transmits it to SpongeBob's Ruby code running on the server. This code turns it into a query to the `Users` table. Considering the "network user" attack model, what attack might SpongeBob's code be vulnerable to?

Command injection XSS **SQL injection** Buffer overflow

E. (2 pts) What was the most challenging bug for you to find and fix in project 7?

Just has to be a bug that you actually had to fix in the project. E.g., buffer overflow would be 0!