Effective Programming in C and Unix - Spring 2011
Written Midterm – 50 points
March 01, 2011

SOLUTION

Name: (please Print) ____

Section (E/F/G) : _______________

Andrew ID: ____________@andrew.cmu.edu

This is a paper and pencil exam. You are allowed to use ONE page of notes. NO notes, program code or compilers allowed. You must not use any computers, mobile phones or PDA's during the test. Please write clearly and legibly. Unclear answers will receive no credit.

	PART 1 – Number Representations	PART 2 – Debugging	PART 3 – Pointers	PART 3 – Linked Lists
MAX POINTS	10	15	10	15
YOUR SCORE				
GRADED BY				

PART I – DATA REPRESENTATIONS (10 pts)

Assume that we have a machine where ints and pointers are represented using 16-bits. Answer the following questions.

1. What is the maximum signed positive integer that can be represented in this machine? Express the answer in hexadecimal only

0X 7FFF

2. What is the minimum signed negative integer that can be represented in this machine? Express the answer in hexadecimal only

0X 8000

3. Show the representation of -2 in this machine (note that the machine does two's compliment arithmetic). Express the answer in hexadecimal only

2 = 0002
~2 = FFFD
+1

1+~2 = FFFE

2=0010
~2=1101
= D

4. Is it true that in this machine we will always get
   a. (x+y)+z = x + (y+z) for all integers x, y and z
   b. x + 1 > x for all integers x. False

If the answer is false (in each case) provide a counter example. If the answer is true (in each case) make a reasonable argument to support your claim.

(1a) true – modula arithmetic preserve mathematical properties

false

(b) X = 7FFF   X+1 = 8000 =

X+1 < X

5. Express 0x45 in decimal

16¹ 16⁰

4x16 + 5 = 69

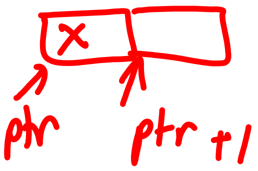
PART II – Debugging Questions (15 points – 3 each)

All questions in this part either seg faults or produce a valid output. If the code is seg faulting, then there is a standard list of error types that you can use to describe the error type. You don't have to use this list though. The list is given below for reference purposes.

A) dereference of uninitialized or otherwise invalid pointer
B) insufficient (or none) allocated storage for operation
C) storage used after free
D) allocation freed repeatedly
E) free of unallocated or potentially storage
F) free of stack space
G) return, directly or via argument, of pointer to local variable
H) dereference of wrong type
I) assignment of incompatible types
J) program logic confuses pointer and referenced type
K) incorrect use of pointer arithmetic
L) array index out of bounds

1. Does this code seg fault or produce an output? Justify in either case.

```
int main(int argc, char* argv[] ){
    int x = 10;
    int* ptr = &x;
    ptr++;
    printf("%x   %d \n",  ptr, *ptr);
}
```



undefined

2. Does this code seg fault or terminate normally with return 0? Explain briefly and point to places in the code that needs to be fixed (if any).

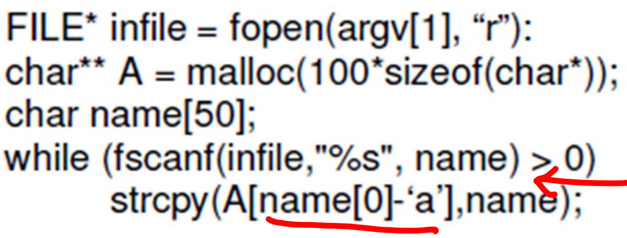
```
#define n 100

int main( ) {
   int** A;
   allocate(A);
   free(A);
   return 0;
}

int allocate(int** array){
      int** arrayint = (int**)malloc(n*sizeof(int*));
      array = arrayint;
      return 0;
   }
```

no memory gets allocated for A

seg fault

local

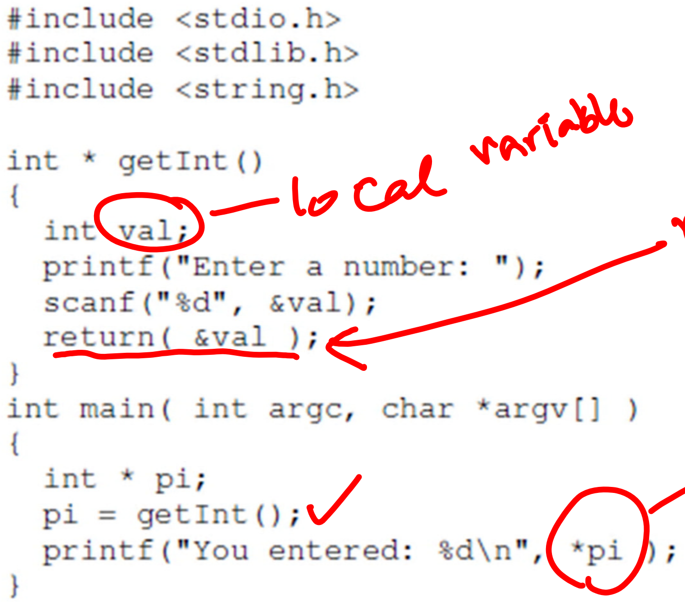
3. What is the purpose of the following code? Will this code seg fault? If so, how would you fix the code to prevent the segmentation fault? Rewrite the code with the fix.



```
FILE* infile = fopen(argv[1], “r”):
char** A = malloc(100*sizeof(char*));
char name[50];
while (fscanf(infile,"%s", name) > 0)
        strcpy(A[name[0]-‘a’],name);
```

malloc space for names

4. Will this code seg fault or produce an output? If output then what is output from this code? If the code seg faults then briefly explain why.



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int * getInt()
{
  int val;
  printf("Enter a number: ");
  scanf("%d", &val);
  return( &val );
}
int main( int argc, char *argv[] )
{
  int * pi;
  pi = getInt();✓
  printf("You entered: %d\n", *pi );
}
```

local variable

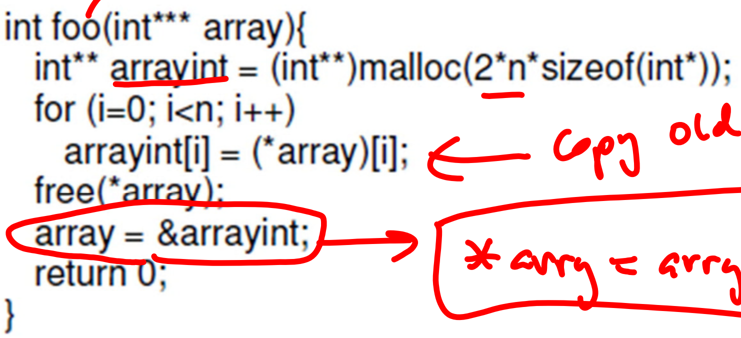
returning an address of a local var

seg fault

5. Study the following code segment. What is the purpose of the foo function? Does it work as intended? If not how would you fix it?

```
int n =10;

int main( ) {
    int** A = malloc(sizeof(int*)*n);
    ……
    foo(&A);
     n *= 2;
    …..
    return 0;
}
```

double the size of array



```
int foo(int*** array){
   int** arrayint = (int**)malloc(2*n*sizeof(int*));
   for (i=0; i<n; i++)
      arrayint[i] = (*array)[i];
   free(*array);
   array = &arrayint;
   return 0;
}
```

copy old

fix

*array = arrayint;

PART III – Pointers (10 points)

1. (5 pts) Consider the following code

A A+5

```
int A[ ]  = {1,2,3,4,5};
int* ptr = A+5;
int* ptr2 = A;
```

ptr

Find the following. For parts where value cannot be determined, you must clearly state why.

(a) *ptr — undefined
(b) *(ptr-3) — 3
(c) ptr – ptr2 — 5
(d) *(ptr – 1) + 2 =7
(e) &ptr = address of variable ptr

A[0] A[1]

2. Consider the following code [3 pts]

int A[2][3] = {{1,2,3},{4,5,6}} and assume that A[0] = 0xFFAA0000

Find the following

Assume (32-bit int)

(a) A[1] = A[0] + 12 C = FFAA000C

(b) *A[1] = A[1][0] = 4

(c) *(A[0] + 1) = = A[0][1] = 2

129
1000 0000

00	00	00	80
		01	00

ptr2

void the question

3. Consider the following code. What are the outputs from the printf statements (2 pts)

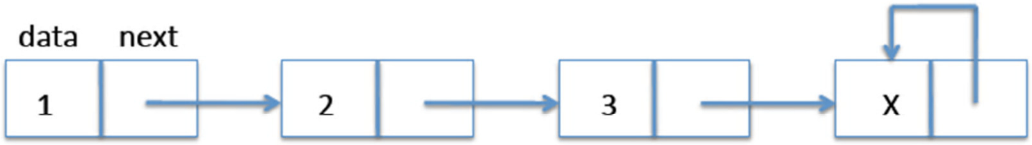
```
int  x = 128;
int* ptr  = &x;
char*  ptr2 = (char*) ptr;
printf("%d  \n",*ptr2);
printf("%d \n",*(ptr2+1));
```

-128

+2 pts given for all exams

PART IV – LINKED LISTS (15 points)

A linked that ends with a self-loop is called a sloop. For example a sloop with 4 nodes is shown below. Note that the last node is a dummy node.



head

A node type of a sloop is defined as

```
typedef struct node {
     int  data;
     struct node* next;
} node;
```

1. Write a function that returns TRUE (defined as 1) if a linked list is a sloop. Otherwise return FALSE (defined as 0) We assume that there are no other cycles in the list and empty list (or one with just the dummy node) is a sloop. You must also write assertions to show assumptions you make.

```
int   is_sloop(node* head ) {
    assert(head != null);  ← ??
    while( head != head->next) {
        head = head->next;
        if (head == null) return false;
    }
    return true;
```

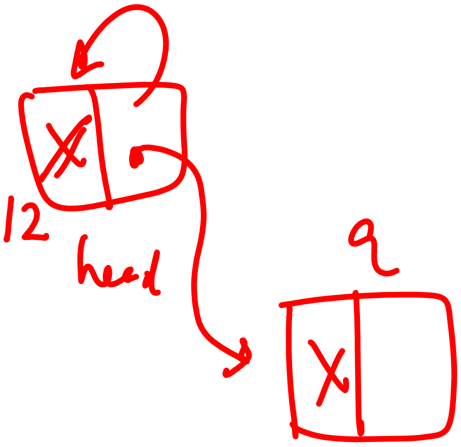
2. The following function addend is supposed to add a node to the end of a sloop. It must retain the sloop property (or invariant). But it contains some bugs. Find the bugs and fix them by rewriting them or adding new statements. Also be sure to write assertions to indicate all your assumptions.

```
node* addend(node* head, int data) {
  node* q = malloc(sizeof(node));
  while ( head != head➔next)
        head = head➔ next;
 head➔data = data;
 head➔next = q;
}
```

node* ptr=head;

q→next=q;

return ptr;



3. Write a function is_equal that takes 2 sloops and returns TRUE (defined as 1) if they have identical nodes and FALSE (defined as 0) otherwise. Be sure to write any assert statements in your code if necessary. Note that two empty sloops are equal by definition.

```
in  is_equal(node* firstsloop,  node* secondsloop) {

  assert(is_sloop(firstsloop) && is_sloop(secondsloop));
  while (firstsloop != firstsloop->next && secondsloop != secondsloop->next){
          firstsloop = firstsloop->next;
          secondsloop = secondsloop->next;
  }
  return (firstsloop == firstsloop->next && secondsloop == secondsloop->next);
```