

CS 60  
Third QUIZ  
July 9, 2009

WRITE ALL YOUR ANSWERS ON SPACE PROVIDED.  
ANSWER ALL FOUR QUESTIONS. TOTAL POINTS IS 33.

NAME: \_\_\_\_\_

**1 { Circle for each part True or False depending whether or not the statement is true or false. Each question is worth 1 Point }**

- { **True** or False } The body of the for statement `for(i = 0; i < 0; i++)` is executed zero times.
- { **True** or False } The String type in C does not exist.
- { True or **False** } When `x` is declared as an integer, then the value of `x` is 2 after executing the statement `x = (5 == 3);`
- { **True** or False } By big-endian we mean that the most significant byte has the largest address byte.
- { **True** or False } The dynamically allocated variables in C are located in an area of memory which in C is called the Heap.
- { True or **False** } The local variable in C are located in an area of memory which in C is called the Heap.
- { True or **False** } The Global and static variables in C are located in an area of memory which in C is called the Heap.
- { True or **False** } A structure (`struct`) in C is an object consisting of two named members of identical types.
- { **True** or False } The function `calloc` returns a pointer to a block of memory all of which has been initialized to zero.
- { **True** or False } A structure (`struct`) in C may contain inside it a union (`union`), and a union (`union`) may contain a structure (`struct`) inside it.

**2 {Pointers}**

(a) [3 points] Briefly explain what happens when the command `pi = &yyy;` is executed. What does the following code print?

```

int xxx = 12;
int yyy = 30;
int *pi = &yyy;
*pi = 31;
pi = &xxx;
printf("%d\n",*pi);           -> 12
pi = &yyy;
*pi 32;
printf("%d %d\n",xxx,yyy);   -> 12 32

```

The address of yyy is stored in pi

(b) [3 points] Briefly explain what happens when the command `*pi = xxx;` is executed. What does the following code print?

```

int xxx = 13;
int yyy = 15;
int *pi = &xxx;
*pi = xxx;
printf("%d\n",*pi);           -> 13
*pi = 52;
printf("%d %d\n",xxx,yyy);   -> 52 15

```

The memory location pointed at by pi gets the value stored at xxx.

(c) [2 points] Briefly explain whether or not the following code generates a segmentation fault. Why or why not? Assume it is the whole program to be run in one of our CSIL machines.

```

int main();
{int *pn;
  pn = (int*) malloc(sizeof(int))
  while(pn)
  { pn = (int*) malloc(sizeof(int));
    *pn = 5;
  }
}

```

-> Segmentation fault

(d) [2 points] Briefly explain whether or not the following `while` loops forever (which is the whole program to be run in one of our CSIL machines)? Why or why not?

```
int main();
{
  int *pn;
  pn = (int *) malloc(sizeof(int));
  while(pn)
  {
    pn = (int *) malloc(sizeof(int));      -> Infinite Loop
    *pn = 5;
    free(pn);
  }
}
```

### 3 {More Questions}

a.- [2 points] For the code given below clearly indicate what the `printf` command prints.

```
int x,y,a,b;
a = 8;
x = ++a;
b = 4;
y = b++;
printf("%d %d %d %d\n", a, b, x, y);      -> 9 5 9 4
```

b.- [3 points] For the code given below clearly indicate what the `printf` command prints.

```
{int *p = (int *) malloc(3*sizeof(int));
  p[0] = 18; p[1] = 15; p[2] = 65;
  printf("%d\n", p[0]);                    -> 18
  p--;
  printf("%d %d\n", p[1], p[2]);          -> 18 15
  p++;
  printf("%d %d\n", p[1], p[2]);          -> 15 65
  free(p);
}
```

## 4 {More Questions}

a.- [4 points] For the code given below clearly indicate what the `printf` command prints.

```
{
  int a;
  int b;
  int *c =&b;
  int *d =&a;
  a = 3; b = 2; *c = 5; *d = 1;
  if (a == b) printf("%d\n", a);
                  else printf("%d\n", b);           -> 5
  if (b/3 == *d) printf("%d\n", a);                 -> 1
                  else printf("%d\n", b);
  if (*c < *d) printf("%d\n", a );
                  else printf("%d\n", b);           -> 5
  if (b = a) printf("%d %d\n", a,b);                -> 1 1
                  else printf("%d %d\n", a,b);
}
```

b.- [4 Points] Below you will find two procedures that are stored in different files which are compiled with the command `gcc proc.c func.c`. Clearly indicate the value(s) they print when we execute the `a.out` executable generated by the above `gcc` command.

`proc.c`

-----

```
#include "stdio.h"
int func(int, int);
int globX = 70;
extern int globCount;

int main(void)
{
    int x=7, y=5, z;
    z = func(y,x);
    printf("%d %d %d\n",z,globX,globCount);    ->  106 71 19
    z = func(globX,globCount);
    printf("%d %d %d\n",z,globX,globCount);    ->  1421 72 18
}
```

`func.c`

-----

```
int globCount = 20;
extern int globX;

int func(int a, int b)
{
    globCount--;
    globX++;
    return(a*b + globX);
}
```