HW 3

Mike Hanling

30 JAN 2016

1. (5 points) Write a small C program that uses `sizeof()` to report the size in byte of each the types listed below. (You don't need to submit the program, just the sizes.)

   (a) `int`

   $\boxed{4}$

   (b) `char`

   $\boxed{1}$

   (c) `int *`

   $\boxed{8}$

   (d) `float *`

   $\boxed{8}$

   (e) `char *`

   $\boxed{8}$

   (f) `short`

   $\boxed{2}$

   (g) `int **`

   $\boxed{8}$

   (h) `float`

   $\boxed{4}$

   (i) `double`

   $\boxed{8}$

2. (5 Points) For the sizes above, why is it that all the pointer types, even the double pointer, have the same size in bytes?

Each of them are pointers, therefore they are the same size. What each *points to* may be different in length, but the actual pointer size is constant.

3. (10 points) Rewrite the following C++ code in C:

```
#include <stdio>
using namespace std;

int main(int argc, char *argv[]){

 int j=10;
 int k;

 cout << "Enter a number" << endl;
 cin >> k;

 cout << "Num+10: " << k + 10 << endl;
}
```

```
#include <stdio.h>

int main(int argc, char *argv[]){

 int j = 10;
 int k;

 printf("Enter a number\n");
 scanf("\%g", &k);|

 printf("Num+10: \%f", k+10);

}
```

4. (10 points) Complete the program below to print ”Go Navy” to a new file called `gonavy.txt`, ”Beat Army” to a `beatarmy.txt`, and ”Crash Airforce” to standard error.

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

int main(int argc, char * argv[]){

   FILE * gonavy, *beatarmy;

   //WRITE THE REST!

}
```

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

int main(int argc, char * argv[]){

   FILE * gonavy, *beatarmy;

   gonavy = fopen("gonavy.txt", "w");
   beatarmy = fopen("beatarmy.txt", "w");

   fprintf(gonavy, "Go Navy");
   fprintf(beatarmy, "Beat Army");
   fprintf(stderr, "Crash Airforce");

   fclose(gonavy);
   fclose(beatarmy);
   return 0;
}
```

5. (10 points) For the following program snippet below, there are at least **five** errors. Enumerate them.

```
for( int i=0 ; i<5 , i--){
   printf(i)
}
```

(a) ; instead of , after `i<5`
(b) **INFINITE LOOP**, maybe change to `i++`
(c) missing first arg to `printf`: "%d"
(d) missing , to separate args of `printf`
(e) missing ; after `printf`

6. (10 points) For the following code snippets, what is the output and explain that output? (Yes, you should program and run these program to determine the output!)

(a)
```
unsigned int i = 4294967295;
printf("%d\n",i);
```

Output: -1
Explanation: unsigned ints go from 0 to 65535. Since this is out of range it yields -1.

(b)
```
int i = 3.1519;
printf("%d\n", i);
```

Output: 3
Explanation: forcing int will simply chop the decimal piece.

(c)
```
int i = (int) 1.5 + 2.5 + 3.5 + 4.5
printf("%d\n",i);
```

Output: 11
Explanation: the `(int) 1.5 + ...` evaluates to `1 + ...` ; adding the remaining floats, the result is 11.5; forcing it to be an int chops the .5, leaving 11.

7. (10 points) Consider the program snippet below and the memory diagram representing that programs state at MARK 0. Complete a stack diagram for each of the remaining MARKS 1-4.

```
int a=0,b=0,*p;
p = &b; /* (0) */

*p = 15; /* (1) */

a = b;

b = 25; /* (2) */

p = &a; /* (3) */

(*p)++; /* (4) */
```

Mark 0 Diagram

```
.----.----.
| a  |  0 |
|----|----|
| b  |  0 | <-.
|----|----|   |
| p  |  .-+---'
'----'----'
```

Mark 1 Diagram

```
.----.----.
| a  |  0 |
|----|----|
| b  |15  | <-.
|----|----|   |
| p  |  .-+---'
'----'----'
```

Mark 2 Diagram

```
.----.----.
```

```
| a  | 15 |
|----|----|
| b  | 25 | <-.
|----|----|   |
| p  |  .-+---'
'----'----'
```

Mark 3 Diagram

```
.----.----.
| a  | 15 | <-.
|----|----|   |
| b  | 25 |   |
|----|----|   |
| p  |  .-+---'
'----'----'
```

Mark 4 Diagram

```
.----.----.
| a  | 16 | <-.
|----|----|   |
| b  | 15 |   |
|----|----|   |
| p  |  .-+---'
'----'----'
```

8. (10 points) What is the values of the array after this code completes?

```
//statically declaring an array
int array[10] = {0,1,2,3,4,5,6,7,8,9};
int * p = array+3;

p[0]=2018;

//<--- Value of array here?
```

{0,1,2,2018,4,5,6,7,8,9}

9. (10 points) You are trying to copy an array from to another, and you write the following code:

```
int a[10] = {0,1,2,3,4,5,6,7,8,9};
int b[10];

//copy from a to b
a=b;
```

(a) Why is this code incorrect?

First, it would say b=a to copy a into b. Second, you cannot set array types equal to each other.

(b) Write a corrected code segment by replacing the offensive part of the code above to copy the values in `b` to `a`.

```
// THIS WILL GIVE RANDOM VALUES TO a because b is uninitialized!!!
for (int i = 0; i < 10; i++){
  a[i] = b[i];
}
```

10. (10 points) The program below has at least three things wrong with them. Enumerate them and write the corrected code.

```
#include <stdlib.h>
int main( int argc, char * argv[]){
  file * stream;

  stream = open("file.txt", "r");

  fprintf(stream, "Hello World");

  return 0;
}
```

(a) `<stdio.h>` not `<stdlib.h>`

(b) FILE not `file`

(c) `fopen` not `open`

(d) `"w"` not `"r"` to be able to print to the stream

```
#include <stdio.h>
int main( int argc, char * argv[]){
  FILE * stream;

  stream = fopen("file.txt", "w");

  fprintf(stream, "Hello World");

  return 0;
}
```