Homework 5

Programming Language Concepts

Due April 17, 2020

Name: Sheng Chen

1. (points) Write a program in the language of your choice that behaves differently if the language used name equivalence than if it used structural equivalence.

In C

```
class Stack {
        int arr[100];
        int number;
        };

 class Queue {
        int arr[100];
        int number;
        };

  Stack x, y;
  Queue r, s;
```

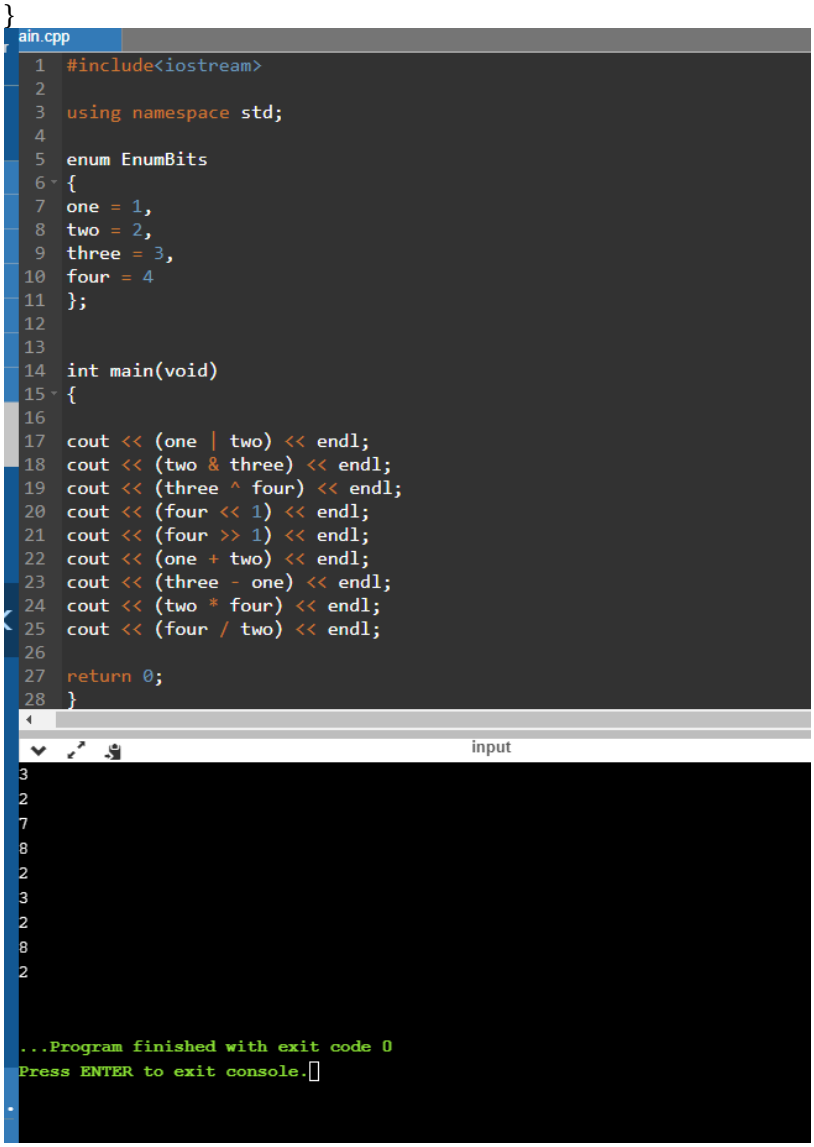
Name equivalence: x and y are same type because their name is equivalence and so does r and s. but, that is false statement for x and r. Structural equivalence: In above example, x and r are structurally equivalent because the class definitions are structurally equivalent Some languages like Java, etc.: still need to check explicit declarations using same rules as for structural.

2. (points) Write a simple program in C++ to investigate the safety of its enumeration types. Include at least 10 different operations on enumeration types to determine what incorrect or just silly things are legal. Now, write a C program that does the same things and run it to determine how many of the incorrect or silly things are legal. Compare your results.

In C++

```
#include<iostream>

using namespace std;

enum EnumBits
{
one = 1,
two = 2,
three = 3,
four = 4
};


int main(void)
{

cout << (one | two) << endl;
cout << (two & three) << endl;
cout << (three ^ four) << endl;
cout << (four << 1) << endl;
cout << (four >> 1) << endl;
cout << (one + two) << endl;
```

```
cout << (three - one) << endl;
cout << (two * four) << endl;
cout << (four / two) << endl;

return 0;
}
```



In C

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

enum Enum
{
   one = 1,
   two = 2,
   three = 3,
   four = 4
};

int main(){
    int intArray[10];
```

```
    printf("%d\n", one | two);
    printf("%d\n", two & three);
    printf("%d\n", three ^ four);
    printf("%d\n", four << 1);
    printf("%d\n", four >> 1);
    printf("%d\n", one + two);
    printf("%d\n", three - one);
    printf("%d\n", two * four);
    printf("%d\n", four / two);

    return 0;
}
```

ain.c

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

enum Enum
{
    one = 1,
    two = 2,
    three = 3,
    four = 4
};

int main(){
    int intArray[10];

    printf("%d\n", one | two);
    printf("%d\n", two & three);
    printf("%d\n", three ^ four);
    printf("%d\n", four << 1);
    printf("%d\n", four >> 1);
    printf("%d\n", one + two);
    printf("%d\n", three - one);
    printf("%d\n", two * four);
    printf("%d\n", four / two);

    return 0;
}
```

input

```
3
2
7
8
2
3
2
8
2


...Program finished with exit code 0
Press ENTER to exit console.
```

Type safety in c++ is less, but it is possible to implicit coneversion between enumerator type and interger, “one op two” when both one and two are enumerator type, int a =enumerator, while C cant doing above three statement with higher type safe.

3. (points) Write a C program that does a large number of references to elements of two- dimensioned arrays, using only subscripting. Write a second program that does the same operations but uses pointers and pointer arithmetic for the storage- mapping function to do the array references. Compare the time efficiency of the two programs. Which of the two programs is likely to be more reliable? Why?

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


int main()
{

clock_t start, end;
double timeUsed;

start = clock();

for(int i=0; i<100000; i++){
subscriptfunc();
}

end = clock();

timeUsed = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("subscript references time used:%f\n",timeUsed);

start = clock();

for(int i=0; i<100000; i++){
pointersfunc();
}

end = clock();

timeUsed = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("pointers time used:%f\n",timeUsed);

return 0;
}

void subscriptfunc()
{
int arr[10][5] = {0};

for(int j=0; j<5 ; j++){
for(int i=0; i<10; i++){
arr[i][j];
}
}

return;
}


void pointersfunc()
{
```

```
int arr[10][5] = {0};
for(int j=0; j<5 ; j++){
for(int i=0; i<10; i++){
*(*(arr+i)+j);
}
}
return;
}
```

```c
#include <stdio.h>
#include <time.h>


int main()
{

clock_t start, end;
double timeUsed;

start = clock();

for(int i=0; i<100000; i++){
subscriptfunc();
}

end = clock();

timeUsed = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("subscript references time used:%f\n",timeUsed);

start = clock();

for(int i=0; i<100000; i++){
pointersfunc();
}

end = clock();

timeUsed = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("pointers time used:%f\n",timeUsed);

return 0;
```

input

```
main.c:38:6: warning: conflicting types for 'subscriptfunc'
main.c:14:1: note: previous implicit declaration of 'subscriptfunc' was here
main.c:52:6: warning: conflicting types for 'pointersfunc'
main.c:26:1: note: previous implicit declaration of 'pointersfunc' was here
subscript references time used:0.010708
pointers time used:0.010197


...Program finished with exit code 0
Press ENTER to exit console.
```

The pointer function is more reliable compared to the subscripting references function because we are accessing the memory directly in pointers function.

4. (points) Analyze and write a comparison of using C++ pointers and Java reference variables to refer to fixed heap- dynamic variables. Use safety and convenience as the primary considerations in the comparison.

C++ pointers have to done allocation manually, and it can point to any variable and any location in memory, but it is much more convenient, because it can assigned any number of times even they are done allocating, arithmetic like addition can be performed on pointers, &p+1;
Java reference are safer than C++ pointers because it refers to class instances which are implicitly de-allocated so that there is dangling references. Since java reference are restrict in its variables, once it was allocated, references can't be reassigned, and arithmetic can't be done with references.