Sheet 03

Ming Qu 2896560

Assignment (Theory) 1

a)
Variable: A portion of memory to store a determined value.
Reference:The address that locates a variable within memory is what we call a reference to that variable.
Pointer: A variable which stores a reference to another variable is called a pointer.

Semantic difference:

- & is the reference operator and can be read as "address of"
- * is the dereference operator and can be read as "value pointed by"

Difference for the compiler:

variable: [type] name; int i = 3;
reference: int &ri = i;
pointer: int *pi = &i;

b)

```
0x7fff5fbff788; 0x7fff5fbff788; 0x7fff5fbff788
10; 10; 10
15; 15; 15
50; 50; 50
10; 10; 10
```

The first line outputs the address of the number 10.
*pointer = 15: set the value pointed by pointer equals to 15.
reference = 50: set the value which address is reference to 10.

c)
callByValue: calling the value itself. value = 0;
callByReference: calling the address of the value. int& reference = value;
callByPointer: calling the pointer of the address of the value. int* pointer = &value, and then set the value pointed by &value equals to 20;

d)

```
//////////////Assignment 1_d ////////////////
10
20
30
60
3.6301e-164
```

set the pointer begin equals to the array
*(begin + 0) equals to array[0], print out the first value of the array
double* end = array + 5; create a new pointer "end" that points to the address that 5 addresses after the array address.

*(begin + 1) = array[1] = 20;
*(end - (diff)) = *(end-2) = *(begin+5-2)=*(begin+3)=40;

There is no undefined behaviour.
e)

set the pointer ptr equals to begin pointer.

10	20	30	40	50

addr(10) *ptr	addr(20) *(ptr+1)	addr(30)	addr(40)	addr(50)	*end	*(end+1)			

The above table illustrates where the ptr and end pointers are. When ptr doesn't point to the address of end pointer, it prints out the value pointed by ptr.

Then, set the pointer ptr equals to the end pointer.

addr(10) *begin	addr(20) *(begin+!)	addr(30)	addr(40)	addr(50) *(ptr-1)	*end *ptr	*(end+1) *(ptr+1)			

Until ptr points to the begin pointer, it prints out the array.

The result is:

```
//////////////Assignment 1_e ////////////////
10 20 30 40 50
50 40 30 20 10
```

f)

```
//////////////Assignment 1_f ////////////////
40 30 20 10
50 40 30 20
50 40 30 20 10
```

The third version works.
Problems:
1st: For the loop"ptr!=begin", it cannot print out *begin.
2nd: first minus ptr, so it cannot print out *(end-1).