# Exercises

## Pointers

Prof. Sai-Keung Wong
TA: xyz

# What you should know…

Please implement the programs to find out the answers on your own.
Don’t trust the answers that are given here.

Some answers are machine dependent!

# Pointer Basics

Write down the meaning of each line.

Explain whether or not each line of code is reasonable. If there is a potential problem, why?

```
int a = 12; int b[] = {1, 2};
int *pa = &a;                    // L0
int *pb = &b;                    // L1
int *pc = b;                     // L2
double *pd = &a;                 // L3
double *pe = (double*)&a;        // L5
int *pf = &b[1];                 // L6
int bb = *(b+1);                 // L7. If no error, what is bb's value?
```

# Pointer Basics: Answers

Write down the meaning of each line.

Explain whether or not each line of code is reasonable. If there is a potential problem, why?

```
int a = 12; int b[] = {1, 2};
int *pa = &a;                  // L0. Assign the address of a to pa
int *pb = &b;                  // L1. Error. b is an address. pb = b is ok
int *pc = b;                   // L2.  pc points to b
double *pd = &a;               // L3. Error. Type mismatch
double *pe = (double*)&a;      // L5. pe points to a. But this is not good.
int *pf = &b[1];               // L6. pf points to the second element of b
int bb = *(b+1);               // L7. bb = b[1] = 2
```

# Pointer Basics

Write down the potential problem of each line (if any). If the line has a problem, the line should be removed.

If there is no problem, write down the meaning of each line and the value of a.

```
int a = 12;
int *p;
*p = 20;                    // L1.

p = &a;                     // L2
*p = *p + 10;               // L3
a = (*p)++ +2;              // L4
a = ++(*p) - 11;            // L5
```

# Pointer Basics: Answers

Write down the potential problem of each line (if any). If the line has a problem, the line should be removed.

If there is no problem, write down the meaning of each line and the value of a.

```
int a = 12;
int *p;
*p = 20;                    // L1. Problem.
                            // It is uncertain which variable that p points to
p = &a;                     // L2   assign the address of a to p. Or p points to a
*p = *p + 10;               // L3                  a = 22
a = (*p)++ +2;              // L4                  a = 25
a = ++(*p) - 11;            // L5                  a = 15
```

**Pointer Basics**

Explain the meaning of L3 step by step.
i.e.,

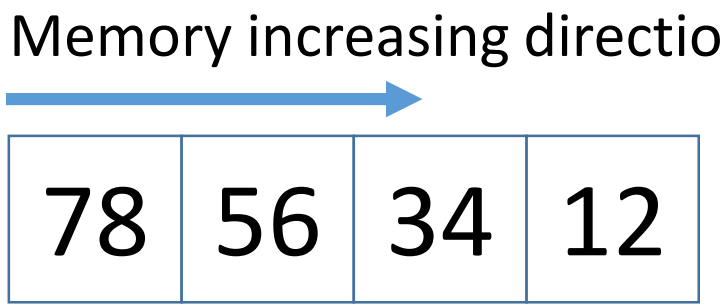
1. (char*)(pCount) ?
2. ((char*)(pCount))[0] ?
3. (int)((char*)(pCount))[0] ?

```
int count = 0x12345678;                          // L1. Assume little endian
int *pCount = &count;                            // L2. What are the purposes?

cout << (int)((char*)(pCount))[0] << endl;       // L3. What is the output?
```

# Pointer Basics: Answers

1. **(char*)(pCount)** ?
   Convert the pointer to point to an array of chars
2. **((char*)(pCount))[0]** ?
   Get the first element of the array pointed by the pointer
3. **(int)((char*)(pCount))[0]** ?
   Convert the element to an integer




```
int count = 0x12345678; // Hexadecimal      // L1. Assume little endian

int *pCount = &count;                       // L2
cout << (int)((char*)(pCount))[0] << endl;  // L3. 0x78 = 120. Output is 120
```

**Pointer Basics**

Explain the meaning of L3 step by step.

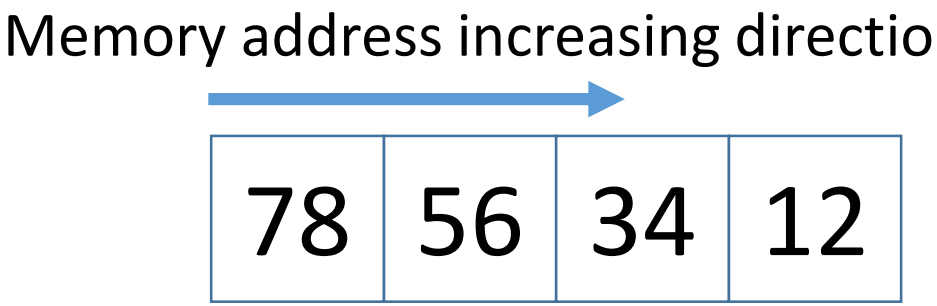
i.e.,

1. (char*)(pCount) ?
2. ((char*)(pCount))[i] ?
3. (int)((char*)(pCount))[i] ?

```
int count = 0x12345678;                          // L1. Little endian
int *pCount = &count;                            // L2
int i; cin >> i;                                 // Input 3
cout << (int)((char*)(pCount))[i] << endl;       // L3. Assume i = 3. Output?
```

## Pointer Basics: Answers

1. **(char*)(pCount)** ?
   Convert the pointer to point to an array of chars
2. **((char*)(pCount))[i]** ?
   Get the (i+1)*th* element of the array pointed by the pointer
3. **(int)((char*)(pCount))[i]** ?
   Convert the element to an integer




```
int count = 0x12345678;                          // L1. Little endian
int *pCount = &count;                            // L2
int i; cin >> i;                                 // input 3
cout << (int)((char*)(pCount))[i] << endl;       // L3. i = 3. output is 18
```

## Pointer Basics

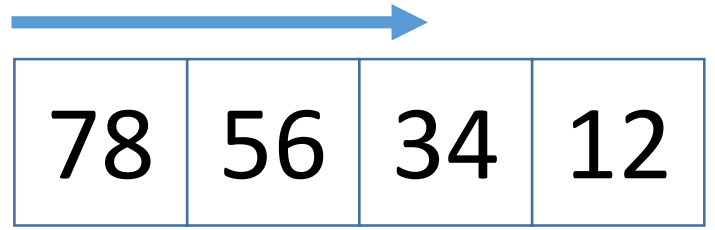
Write down the purpose(s) of each line.
Assumption: little endian for data storage
What are the output?

```
int count = 0x12345678;                                          // L1
int *pCount = &count;                                            // L2
cout << hex << endl;
cout << "pCount:" << pCount << endl;                             // L3
cout << "*pCount:" << *pCount << endl;                           // L4

cout << "1st byte:"  << (int)((char*)(pCount))[0] << endl;       // L5
cout << "2nd byte:" << (int)((char*)(pCount))[1] << endl;        // L6
cout << "3rd byte:"  << (int)((char*)(pCount))[2] << endl;       // L7
cout << "4th byte:"  << (int)((char*)(pCount))[3] << endl;       // L8
```

# Pointer Basics : Answers

Write down the purpose(s) of each line.
Assumption: little endian for data storage
What are the output?




```
int count = 0x12345678;                          // L1
int *pCount = &count;                            // L2
cout << hex << endl;                             // output in hexadecimal
cout << "pCount:" << pCount << endl;             // L3. Address of count
cout << "*pCount:" << *pCount << endl;           // L4. 12345678

cout << "1st byte:"   << (int)((char*)(pCount))[0] << endl; // L5.   78
cout << "2nd byte:" << (int)((char*)(pCount))[1] << endl;  // L6.   56
cout << "3rd byte:"  << (int)((char*)(pCount))[2] << endl;  // L7.   34
cout << "4th byte:"  << (int)((char*)(pCount))[3] << endl;  // L8.   12
```

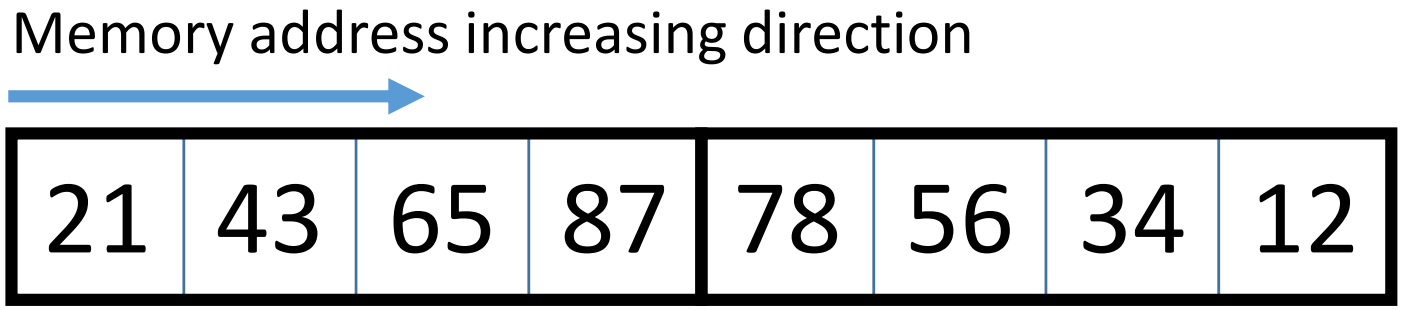
# Pointer Basics

Explain the meaning of each line in a step-by-step manner. And write down the output if any.

```
int a[] = {0x87654321, 0x12345678};   // L1. Little endian
int *p = a;                           // L2
cout << hex << endl;                  // output in hex
cout << p[0] << endl;                 // L3.
cout << (p+1)[0] << endl;             // L4
cout << (int)(((char*)p)+1)[3] << endl; // L5
cout << *(p+1) << endl;               // L6
cout << ((short*)p+1)[0] << endl;     // L7
cout << ((short*)p+2)[1] << endl;     // L8
```

# Pointer Basics: Answers. Step 1

Explain the meaning of each line in a step-by-step manner. And write down the output if any.

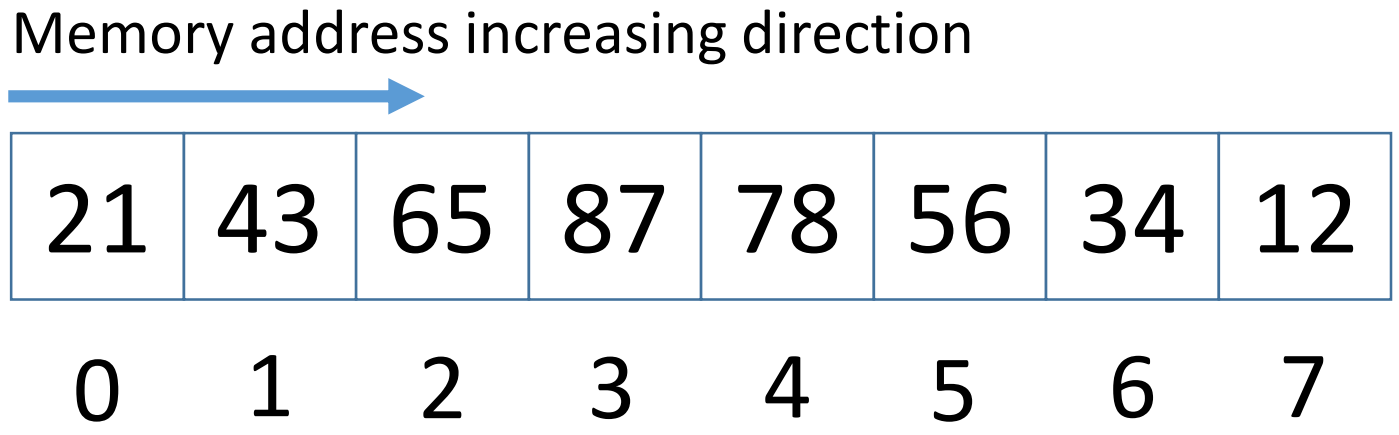



Four bytes per element

```
int a[] = {0x87654321, 0x12345678};          // L1. Little endian
int *p = a;                                  // L2
cout << hex << endl;                         // output in hex
cout << p[0] << endl;                        // L3.        87654321
cout << (p+1)[0] << endl;                    // L4.        12345678
```

**Pointer Basics: Answers.** **Step 2**

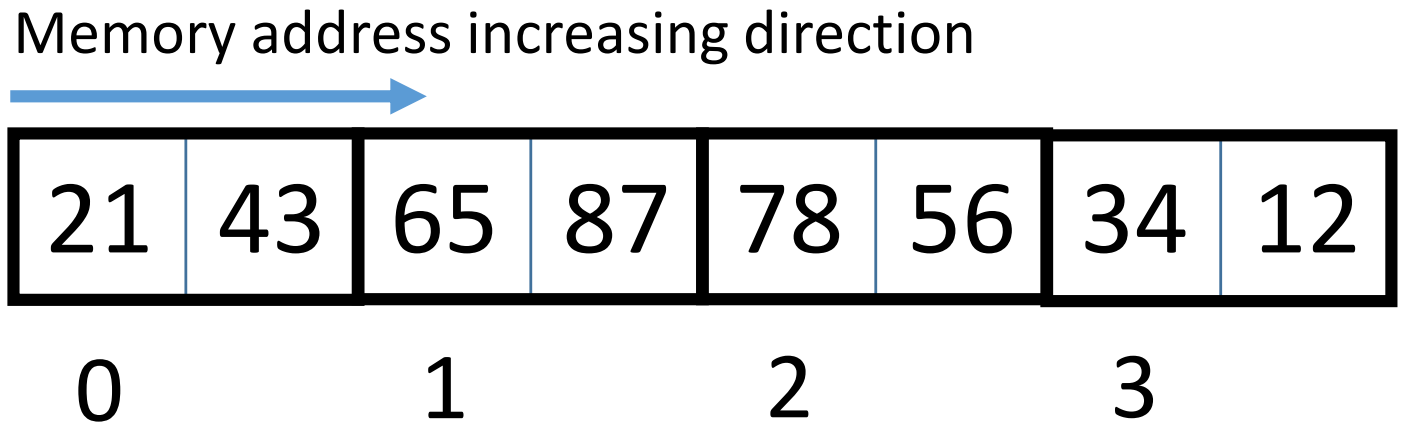
Explain the meaning of each line in a step-by-step manner. And write down the output if any.




```
int a[] = {0x87654321, 0x12345678};          // L1. Little endian
int *p = a;                                  // L2
cout << hex << endl;                         // output in hex
……
cout << (int)(((char*)p)+1)[3] << endl;      // L5.        78
cout << *(p+1) << endl;                      // L6.        12345678
```

**Pointer Basics: Answers.** **Step 3**

Explain the meaning of each line in a step-by-step manner. And write down the output if any.




```
int a[] = {0x87654321, 0x12345678};          // L1. Little endian
int *p = a;                                  // L2
cout << hex << endl;                         // output in hex
……
cout << ((short*)p+1)[0] << endl;            // L7.        8765
cout << ((short*)p+2)[1] << endl;            // L8.        1234
```

**Pointer Basics**

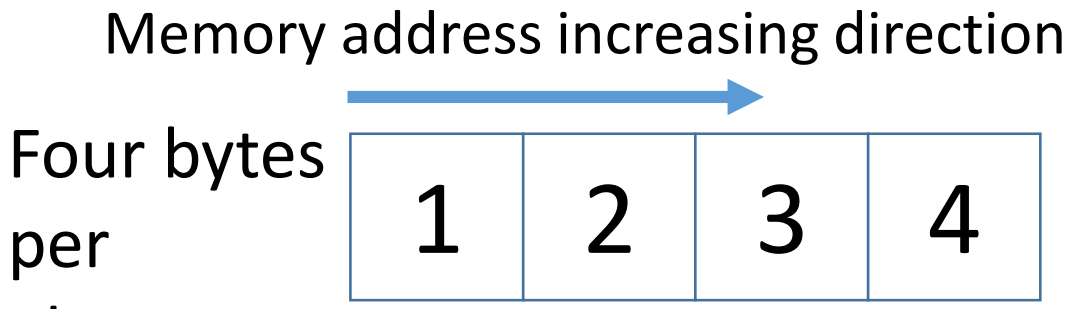
# What are the output?

```
int counts[] = {1, 2, 3, 4};
int *pCount = counts;                                   //

cout << (pCount+0)[0] << endl;                          // L1
cout << (pCount+1)[0] << endl;                          // L2
cout << (pCount+2)[0] << endl;                          // L3
cout << (pCount+3)[0] << endl;                          // L4
```

**Pointer Basics: Answers**

# What are the output?




```
int counts[] = {1, 2, 3, 4};
int *pCount = counts;                                   //

cout << (pCount+0)[0] << endl;                          // L1.      1
cout << (pCount+1)[0] << endl;                          // L2.      2
cout << (pCount+2)[0] << endl;                          // L3.      3
cout << (pCount+3)[0] << endl;                          // L4.      4
```

# Pointer Basics

What are the output?

```
int a = 12, b[] = {4, 5, 6, 7};
int *p = b;
int *pa = &a;
b[3] = (*p+1) + b[0];
b[0] = *pa + *p;
*pa = b[1] + b[3];
pa = b;
b[2] = b[0] + pa[2];
cout << a << “\t” << b[0] << “\t” << b[1]  << “\t” << b[2] << “\t” << b[3];
```

## Pointer Basics: Answers. Step 0

What are the output?

```
int a = 12, b[] = {4, 5, 6, 7};
int *p = b;
int *pa = &a;
b[3] = (*p+1) + b[0];                ;
b[0] = *pa + *p;                     ;
*pa = b[1] + b[3];                   ;
pa = b;                              ;
b[2] = b[0] + pa[2];                 ;
cout << a << "\t" << b[0] << "\t" << b[1]  << "\t" << b[2] << "\t" << b[3];
```

**14 16 5 22 9**

# Pointer Basics: Answers. Step 1

What are the output?

```
int a = 12, b[] = {4, 5, 6, 7};
int *p = b;
int *pa = &a;
b[3] = (*p+1) + b[0];          ; b[3] = (b[0] + 1 ) + b[0] = 4+1+4 = 9
b[0] = *pa + *p;               ;
*pa = b[1] + b[3];             ;
pa = b;                        ;
b[2] = b[0] + pa[2];           ;
cout << a << "\t" << b[0] << "\t" << b[1]  << "\t" << b[2] << "\t" << b[3];
```

14 16 5 22 9

## Pointer Basics: Answers. Step 2

What are the output?

```
int a = 12, b[] = {4, 5, 6, 7};
int *p = b;
int *pa = &a;
b[3] = (*p+1) + b[0];          ; b[3] = (b[0] + 1 ) + b[0] = 4+1+4 = 9
b[0] = *pa + *p;               ; b[0] = a + b[0] = 12 + 4 = 16
*pa = b[1] + b[3];             ;
pa = b;                        ;
b[2] = b[0] + pa[2];           ;
cout << a << "\t" << b[0] << "\t" << b[1]  << "\t" << b[2] << "\t" << b[3];
```

14 16 5 22 9

## Pointer Basics: Answers. Step 3

What are the output?

```
int a = 12, b[] = {4, 5, 6, 7};
int *p = b;
int *pa = &a;
b[3] = (*p+1) + b[0];          ; b[3] = (b[0] + 1 ) + b[0] = 4+1+4 = 9
b[0] = *pa + *p;               ; b[0] = a + b[0] = 12 + 4 = 16
*pa = b[1] + b[3];             ; a = b[1] + b[3] = 5 + 9 = 14
pa = b;                        ;
b[2] = b[0] + pa[2];           ;
cout << a << "\t" << b[0] << "\t" << b[1]  << "\t" << b[2] << "\t" << b[3];
```

14 16 5 22 9

## Pointer Basics: Answers. Step 4

What are the output?

```
int a = 12, b[] = {4, 5, 6, 7};
int *p = b;
int *pa = &a;
b[3] = (*p+1) + b[0];          ; b[3] = (b[0] + 1 ) + b[0] = 4+1+4 = 9
b[0] = *pa + *p;               ; b[0] = a + b[0] = 12 + 4 = 16
*pa = b[1] + b[3];             ; a = b[1] + b[3] = 5 + 9 = 14
pa = b;                        ;
b[2] = b[0] + pa[2];           ; b[2] = b[0] + b[2] = 16 + 6 = 22
cout << a << “\t” << b[0] << “\t” << b[1]  << “\t” << b[2] << “\t” << b[3];
```

14 16 5 22 9

# Pointer Basics

```
int *p = _______;   // allocate a memory space for an integer
                    // and assign its address to p
int *pa  = _____;   // allocate a memory space for 128 integers
                    // and assign the address to pa
______;             // delete the memory space pointed by p
______;             // delete the memory space pointed by pa
```

# Pointer Basics: Answers

```
int *p = new int;      // allocate a memory space for an integer
                       // and assign its address to p
int *pa  = new int[128];   // allocate a memory space for 128 integers
                           // and assign the address to pa
delete p;              // delete the memory space pointed by p
delete [] pa;          // delete the memory space pointed by pa
```

# Pointer Basics

```
double*pa;

….

// assume that pa may point to a memory space which stores a set of double numbers

// Write a piece of code to delete the memory space pointed by pa

// if pa is not NULL. Also, set pa to a proper value.
```

# Pointer Basics: Answers

```
double*pa;

….

// assume that pa may point to a memory space which stores a set of double numbers

// Write a piece of code to delete the memory space pointed by pa

// if pa is not NULL. Also, set pa to a proper value.


if (pa) { delete [] pa; pa = NULL; }
```

# Pointer Basics

// allocate an array of 128 pointers to arrays of double numbers

double**pa = _____;

# Pointer Basics: Answers

```
// allocate an array of 128 pointers to arrays of double numbers
double**pa = new double*[128];
```

# Pointer Basics

// allocate an array of 128 pointers to arrays of double numbers.

// The size of each array of double numbers is 16.

double**pa = _____;

_____

_____

……

## Pointer Basics: Answers. Step 0

```
// allocate an array of 128 pointers to arrays of double numbers.
// The size of each array of double numbers is 16.

double**pa = new double*[128];
for (int i = 0; i < 128;++i) {
        pa[i] = new double[16];
}
```

## Pointer Basics: Answers. Step 1

```
// allocate an array of 128 pointers to arrays of double numbers.
// The size of each array of double numbers is 16.

double**pa = new double*[128];
for (int i = 0; i < 128;++i) {
        pa[i] = new double[16];
}

// pa is a two dimensional array!
// What does pa[2][15] mean?
```

## Pointer Basics: Answers. Step 2

```
// allocate an array of 128 pointers to arrays of double numbers.
// The size of each array of double numbers is 16.

double**pa = new double*[128];
for (int i = 0; i < 128;++i) {
        pa[i] = new double[16];
}
```

**// pa is a two dimensional array!**
**// What does pa[2][15] mean?**
**// p[2] is the pointer to the array of double numbers**
**// p[2] is the third array of double numbers**
**// p[2][15] is the 16th element of the third array**

# Pointer Basics

Implement a function foo to return a memory space which stores n integers.

# Pointer Basics: Answer

Implement a function foo to return a memory space which stores n integers. There are many ways to define foo.

```
int *foo(int n) {
        if (n <=0) return 0;
        return new int[n];
}
```

```
int *foo(int n) {
        if (n <=0) return 0;
        int *p = new int[n];
        return p;
}
```

# Pointer Basics

Implement a function foo to allocate two memory spaces which store n integers and m double numbers, respectively. Use pass-by-reference.

Receive: n, m, vi and vd. vi and vd are pointers to a pointer.

vi points to a pointer which points to a set of integers

vd points to a pointer which points to a set of double numbers

Return: return the addresses to vi and vd

```
void foo( ……) {
        ……
}
```

# Pointer Basics: Answers

Implement a function foo to allocate two memory spaces which store n integers and m double numbers, respectively. Use pass-by-reference.

Receive: n, m, vi and vd. vi and vd are pointers to a pointer.

vi points to a pointer which points to a set of integers

vd points to a pointer which points to a set of double numbers

Return: return the addresses to vi and vd

```
void foo( int n, int m, int **vi, double **vd){
        if (n<=0) *vi = 0; else *vi = new int[n];
        if (m<=0) *vd = 0; else *vd = new double[m];
}
```