Problem V: Alignment (15pts)

a) (4pts) What is the **maximum** alignment in Bytes of the following addresses:

- 0XCBB7:
- 0XCB10:
- 0xCB60:
- 0xCB88:

b) (6pts) What is the ***Total Size*** of each of the structures{A,B,C} and ***show padding*** in table form?

- Do not reorder

Standard Sizes:

char	- 1 byte
int	- 4 bytes
float	- 4 bytes
float *	- 4 bytes
double	- 8 bytes
Vect4D_SIMD	- 16 bytes

```
struct A
{
        char    a;
        int     j;
        int     k;
        char    b;

}
```

```
struct B
{
        char    c;
        char    d;
        double  r;
        float   m;
};
```

```
struct C
{
        A       d_A;
        B       d_B;
        int     *p;
        double  s;
        float   n;
};
```

c) (5pts) Please write the new ***struct D*** below. Assume that ***struct D*** is instantiated at address 0x0.

- Needs to match the compiler's padding layout

```
struct D
{
    A           d_A;
    B           d_B;
    Vect4D_SIMD v_C;
};
```

i. Add padding to ***struct D***, so that the address ***v_C*** is on a ***16-byte*** boundary.
   - You ***CANNOT*** change ***struct A*** or ***struct B*** at all.
   - You ***CANNOT*** change the order of variables in ***struct D,*** only the padding in ***struct D.***

ii. Please show all the padding in ***struct D*** to guarantee alignment of address ***v_C*** is on a ***16-byte*** boundary.

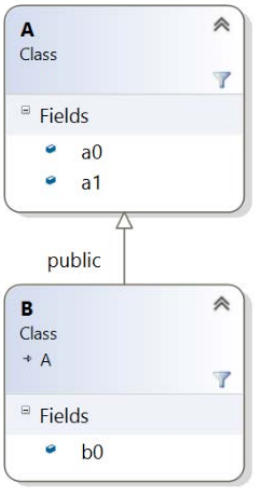
Problem W: Pointers (as promised) (10pts)

Assume that we are working on a ***LITTLE*** endian processor

```
unsigned char data[] =   // Addresses italicized
{
      0x0000:  0xEF, 0xCF, 0x12, 0x90,
      0x0004:  0xA8, 0xB7, 0xE3, 0x63,
      0x0008:  0x64, 0xAA, 0x31, 0xFF,
      0x000C:  0xED, 0x15, 0x0B, 0xAD,
      0x0010:  0x99, 0x05, 0x85, 0x44,
      0x0014:  0xBB, 0x55, 0x78, 0x12,
      0x0018:  0x77, 0xA8, 0x9B, 0xCC,
      0x001C:  0xCE, 0x89, 0x66, 0x27,
      0x0020:  0xC9, 0x18, 0xB3, 0xE7
};

unsigned char  *p;  // char are 8-bits wide
unsigned int   *r;  // ints are 32-bits wide
unsigned short *s;  // shorts are 16-bits wide

p = &data[0];                                  Expected Output

Trace::out("%x\n",  p[0]   );      1) _____________________________

p = p + 7;

Trace::out("%x\n",  p[-2]  );      2) _____________________________

Trace::out("%x\n",  *p++   );      3) _____________________________

Trace::out("%x\n",  *(++p) );      4) _____________________________

p += 5;

Trace::out("%x\n",  *(3+p) );      5) _____________________________

Trace::out("%x\n",  *(p++) );      6) _____________________________

p = (p + 5);

Trace::out("%x\n",  *--p   );      7) _____________________________

Trace::out("%x\n",  *(p+2) );      8) _____________________________

--p;

Trace::out("%x\n",  *p--   );      9) _____________________________

p = p + 3;

Trace::out("%x\n",  p[1]   );     10) _____________________________
```

```
unsigned char data[] =   // Addresses italicized
{
      0x0000:  0xEF, 0xCF, 0x12, 0x90,
      0x0004:  0xA8, 0xB7, 0xE3, 0x63,
      0x0008:  0x64, 0xAA, 0x31, 0xFF,
      0x000C:  0xED, 0x15, 0x0B, 0xAD,
      0x0010:  0x99, 0x05, 0x85, 0x44,
      0x0014:  0xBB, 0x55, 0x78, 0x12,
      0x0018:  0x77, 0xA8, 0x9B, 0xCC,
      0x001C:  0xCE, 0x89, 0x66, 0x27,
      0x0020:  0xC9, 0x18, 0xB3, 0xE7
};

r = (unsigned int *)&data[0];                Expected Output

Trace::out("%x\n",  r[1]   );      11) _____________________________

r += 3;

Trace::out("%x\n",  *(r-1) );      12) _____________________________

r++;

s = (unsigned short *)r;

Trace::out("%x\n",  s[0]   );      13) _____________________________

s += 2;

Trace::out("%x\n",  *(3+s) );      14) _____________________________

Trace::out("%x\n",  *s++   );      15) _____________________________

s += 3;

p = (unsigned char *)s;

Trace::out("%x\n",  *++p   );      16) _____________________________

p -= 10;

Trace::out("%x\n",  *p++   );      17) _____________________________

Trace::out("%x\n",  p[3]   );      18) _____________________________

p += 5;

s = (unsigned short *)p;

Trace::out("%x\n",   *--s   );      19) _____________________________

r = (unsigned int *)s;

Trace::out("%x\n",  *(++r) );      20) _____________________________
```

Problem X: Class Tracing [code] (10pts)




```
class A
{
public:
A()
: a0(5), a1(7)
{
    Trace::out("A-label: 1111 \n");
}

A(const A & tmp)
: a0(tmp.a0), a1(tmp.a1)
{
    Trace::out("A-label: 1122 \n");
}

A(int v0, int v1)
:a0(v0), a1(v1)
{
    Trace::out("A-label: 1133 \n");
}

A & operator = (const A &tmp)
{
    this->a0 = tmp.a0;
    this->a1 = tmp.a1;
    Trace::out("A-label: 1144 \n");
    return *this;
}

~A()
{
    Trace::out("A-label: 1155 \n");
}

A operator + (const A &tmp)
{
    A result;
    result.a0 = this->a0 + tmp.a0;
    result.a1 = this->a1 + tmp.a1;
    Trace::out("A-label: 1166 \n");
    return result;
}

// data
int a0;
int a1;
};
```

```
class B : public A
{
public:
B()
    : b0(55)
{
    Trace::out("B-label: 2211 \n");
}

B(const B & tmp)
    : b0(tmp.b0)
{
    Trace::out("B-label: 2222 \n");
}

B(int v0, int v1, int v2)
    :A(v0,v1), b0(v2)
{
    Trace::out("B-label: 2233 \n");
}

B & operator = (const B &tmp)
{
    this->b0 = tmp.b0;
    // a is handled in base class
    Trace::out("B-label: 2244 \n");
    return *this;
}

~B()
{
    Trace::out("B-label: 2255 \n");
}

B operator + (const B &tmp)
{
    B result;
    result.b0 = this->b0 + tmp.b0;
    // a is handled in base class
    Trace::out("B-label: 2266 \n");
    return result;
}

B operator - (const B &tmp)
{
    Trace::out("B-label: 2277 \n");
    return B(this->a0 - tmp.a0,
          this->a1 - tmp.a1,
          this->b0 - tmp.b0);
}

// data
int b0;
};
```

Example Trace:

Code:

```
B B9;
```

Trace: 1111 *As you walk through the code (Tracing),*
2211 *list the labels in the order they are called.*

a) (1pt) What is the Trace of the following chunk of code:

```
B B4(2, 4, 6);
B B5 = B4;
```

Trace:

b) (1pt) What is the Trace of the following chunk of code:

```
B B0;
B B1(3, 6, 9);
B0 = B1;
```

Trace:

c) (2pts) What is the Trace of the following chunk of code:

```
B *pB = new B(2,4,6);
delete pB;
```

Trace:

d) (2pts) What is the Trace of the following chunk of code:

```
A *pA = new B(2,4,6);
delete pA;
```

Trace:

e) (2pts) What is the Trace of the following chunk of code: (assume B1, B4 already constructed)

```
B B6 = B1 + B4;
```

Trace:

f) (2pts) What is the Trace of the following chunk of code: (assume B1, B4 already constructed)

```
B B7 = B1 - B4;
```

Trace:

Problem Y: Memory Leaks [code] (10 pts)

- Rework the CLASSES to prevent MEMORY LEAKS
  - Do not change anything in main()
- For small modifications to existing methods
  - Cross the method out and rewrite on next page
- For a small signature change (i.e. adding const) just overwriting existing method
  - Make sure it easy to see your addition


```
int main()
{
   Puppy *pA = new Puppy(Puppy::Name::Aristotle);
   Puppy *pB = new Puppy(Puppy::Name::Plato);
   Puppy *pC = new Puppy(Puppy::Name::Pythagoras);
   Puppy *pD = new Puppy(Puppy::Name::Socrates);

   Dog *pDog = new Akita(Akita::Color::Brindle,
                         Dog::Type::Working,
                         100.0f);

   pDog->Add(pA);
   pDog->Add(pB);
   pDog->Add(pC);
   pDog->Add(pD);

   pDog->PuppyPrint();

   delete pDog;
}
```

```
class Puppy
{
public:
   enum class Name
      {
         Plato,
         Socrates,
         Aristotle,
          Pythagoras
      };

   const char *stringName[4]
      {
         "Plato     ",
         "Socrates  ",
         "Aristotle ",
         "Pythagoras"
      };

public:
   Puppy(Name n)
      {
         this->mName = n;
         this->pNext = nullptr;
       }

    Puppy *pNext;
    Name  mName;
};

class Dog
{
public:
   enum class Type
      {
         Working,
         Herding,
         Sport,
         Hound
       };

   virtual void PuppyPrint() = 0;
   virtual void Add(Puppy *pPuppy) = 0;

public:
       Type mType;
       float mWeight;
};
```

```
class Akita : public Dog
{
public:
       enum class Color
       {
          Brindle,
          White,
          Brown,
          Black
       };

public:
     Akita(Color c, Type t, float weight)
       {
          this->mColor = c;
          this->mType = t;
          this->mWeight = weight;
          this->p = nullptr;
       }

       void Add(Puppy *pPuppy)
       {
          if (p != nullptr)
          {
              pPuppy->pNext = this->p;
          }
          this->p = pPuppy;
       }

       void PuppyPrint() override
       {
          Puppy *pTmp = this->p;
          while (pTmp != 0)
             {
              Trace::out("Name: %s\n",
              pTmp->stringName[(int)
                           pTmp->mName]);
              pTmp = pTmp->pNext;
              }
       }

private:
       Puppy *p;
       Color mColor;
};
```

Problem Z: Applying what you know [code] (hardest problem) (10 pts)
Refactor Work class to add a proxy function to compare the years of experience to get the desired output. *See test code below*

```
// ---------------------------------------------------------------------------------------------------------------------------
//  Please REFACTOR Work class, feel free to add/delete/modify any method.
//   Add any necessary Proxy structures/classes to accomplish the goal
//   NOTE: you can change  ANYTHING in this class (hint)
// ---------------------------------------------------------------------------------------------------------------------------
```

```
class Work
{
public:
	enum class Name
	{
	   Ed,
	   Sara
	};


public:
	Work(Name name, int yearsExperience)
	{
	   this->mName = name;
	   this->mYears = yearsExperience;
	}

	int GetYears()
	{
	   return this->mYears;
	}

	const char *GetName()
	{
	   return this->StringName[(int)this->mName];
	}


private:
	const char *StringName[2]
	{
	  "Ed",
	  "Sara"
	};

private:
	int  mYears;
	Name mName;

};
```

```
// ------------------------------------------------------------------------------------------------------------------------
// DO NOT CHANGE anything below this line, this is the test function, should work as is, leave it alone.
// ------------------------------------------------------------------------------------------------------------------------

int main()
{
  Work A(Work::Name::Ed, 26);
  Work B(Work::Name::Sara, 2);

  // Ed - 26 years
  Trace::out("\n");
  Trace::out("% 5s --> % 3d years of experience \n", A.GetName(), A.GetYears());

  // Sara - 2 years
  Trace::out("% 5s --> % 3d years of experience \n", B.GetName(), B.GetYears());
  Trace::out("\n");

  // Who is better based on years?
  if ( A.GetYears() > B.GetYears() )
  {
     Trace::out("%s is a better programmer than %s \n", A.GetName(), B.GetName() );
  }
  else
  {
     Trace::out("%s is a better programmer than %s \n", B.GetName(), A.GetName());
  }
}

// ------------------------------------------------------------------------------------------------------------------------
// DESIRED OUTPUT Refactor Work class and add proxy to get desired results
// ------------------------------------------------------------------------------------------------------------------------

          Ed -->  26 years of experience
        Sara -->   2 years of experience

       Sara is a better programmer than Ed
```