Programming Study
Pointer

Sungwoo Nam

2019.3.11

Pointer Usage

```
int a = 1234;
int *pa = &a;     // pa is pointer of a
*pa = 1111;       // assignment on *pa will change a also

int &ra = a;      // ra is reference of a
ra = 4321;        // assignment on ra will change a also
```

Function Return using reference and pointer

```
int AddAndMultiply(int a, int b, int& mult)
{
  mult = a * b;
  return a + b;
}

int AddAndMultiplyV3(int a, int b, int* mult)
{
  *mult = a * b;
  return a + b;
}


int mult = 0;
int sum = AddAndMultiply(1, 2, mult);
sum = AddAndMultiplyV3(1, 2, &mult);
```

Function Return using struct

```
struct AddMult
{
  int Add;
  int Mult;
};

AddMult AddAndMultiplyV3(int a, int b)
{
  AddMult r;
  r.Add = a + b;
  r.Mult = a * b;
  return r;
}

AddMult r = AddAndMultiplyV3(1, 2);
sum = r.Add;
mult = r.Mult;
```

Const Usage

```
const int* cpa = &a;
*cpa = 42;            // compile error : cpa is const pointer
cpa = &b;             // but you can make it point others

const int& cra = a;
cra = 42;             // compile error : cra is const reference

int* const pca = &a;
*pca = 42;            // no problem
*pca = &b;            // compile error : pointer itself is const
```

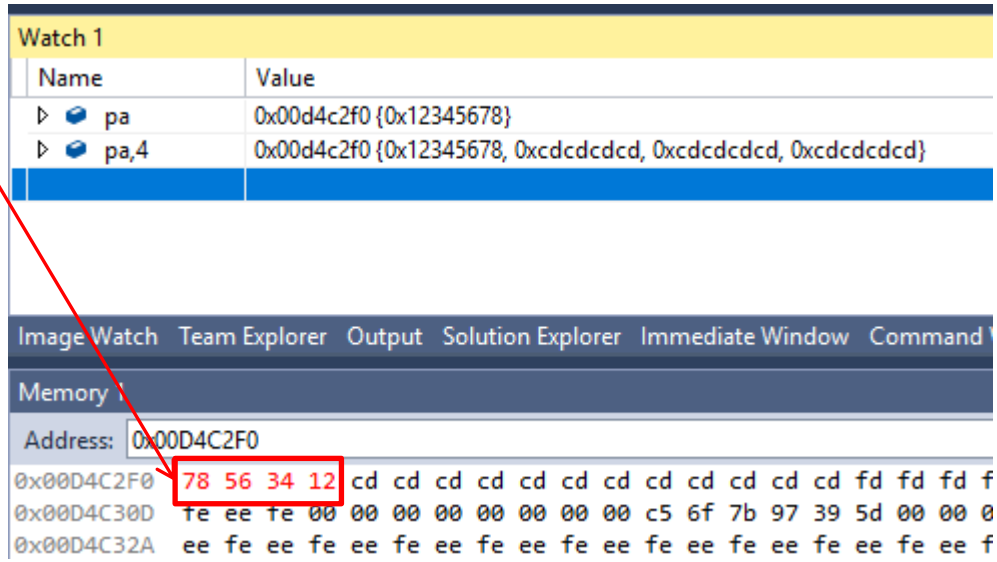
a	42
pa	0x00b9f4a0 {42}
ra	42
b	2222
cpa	0x00b9f48c {2222}
cra	42
pca	0x00b9f4a0 {42}

Const on class method

```
struct AddData
{
  int A;
  int B;
  mutable int C = 0;  // ok to be changed on const method

  int AddAll() const  // This doesn’t change member variable
  {
    if (C == 0) { C = A + B; }
    return C;
  }
};
```

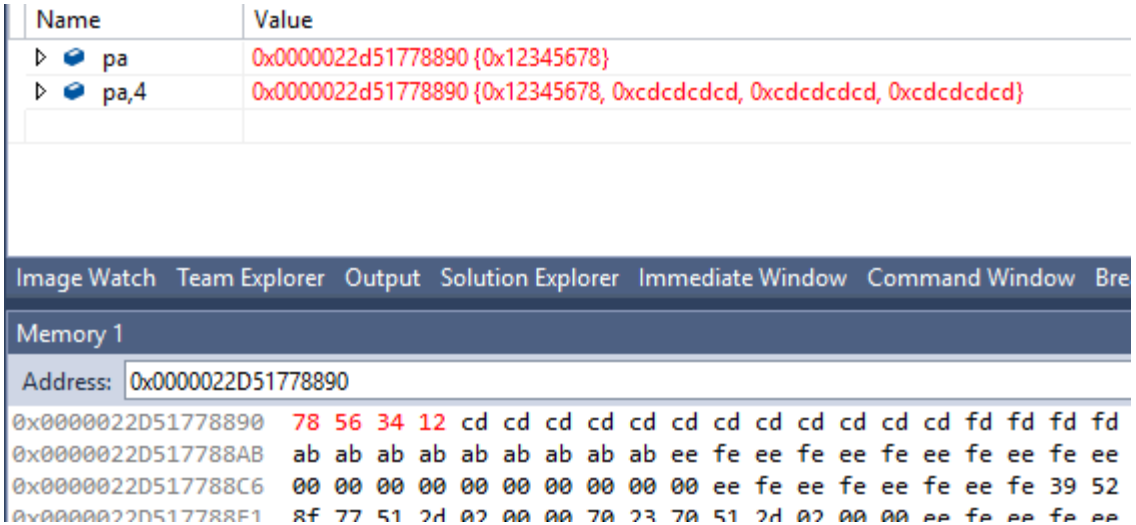
Endianess

```
uint32_t *pa = new uint32_t[4];
pa[0] = 0x12345678;     // note the little-endian
pa[1] = 0x87654321;
pa[2] = 0x10101010;
pa[3] = 0xCAFEBEEF;

*pa = 0xFFFFFFFF;
*(pa + 1) = 0xEEEEEEEE;

delete[] pa;
```



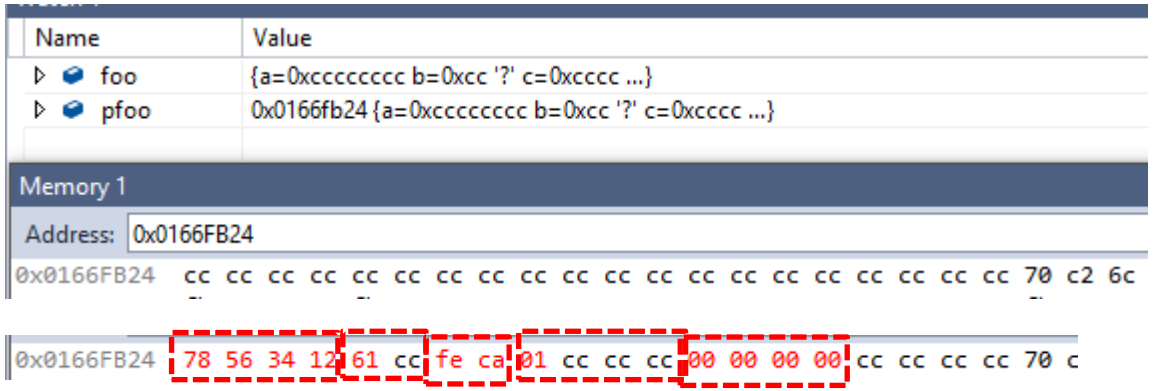
x64 application

```
uint32_t *pa = new uint32_t[4]; // note pa is 64 bit length
pa[0] = 0x12345678;
pa[1] = 0x87654321;
pa[2] = 0x10101010;
pa[3] = 0xCAFEBEEF;

*pa = 0xFFFFFFFF;
*(pa + 1) = 0xEEEEEEEE;

delete[] pa;
```



Struct pack

```
struct Foo
{
  uint32_t a;      // 4 Bytes
  char b;          // 2 Bytes
  uint16_t c;      // 2 Bytes
  bool d;          // 4 Bytes
  const char * e; // 4 Bytes
};

Foo foo;
Foo* pfoo = &foo;
foo = { 0x12345678, 'a', 0xCAFE, true, nullptr };
```



Function Pointer

```
int TestBar(char a) {  return a + 42;  }
int TestToo(char a) {  return a + 22;  }

typedef int(*FooFn)(char);
FooFn f = TestBar;
int ret = f('a');
f = TestToo;
ret = f('b');
```

Function Pointer using STL

```
int TestBar(char a) {  return a + 42;  }
int TestToo(char a) {  return a + 22;  }

using FooFn = std::function<int(char)>;
FooFn fn = TestBar;
int ret = fn('a');
fn = TestToo;
ret = fn('b');
```