POINTERS

IT104021 - COMPUTER PROGRAMMING

NISHANTHA ANURUDDHA

OUTLINE

- Introduction to Pointers
- Declaring a Pointer
- Initializing a Pointer
- Dereference operator
- Null Pointer
- Pointer Operations
- Pointers and Arrays
- Pointer vs Reference
- Pointer to Pointer
- Exercise

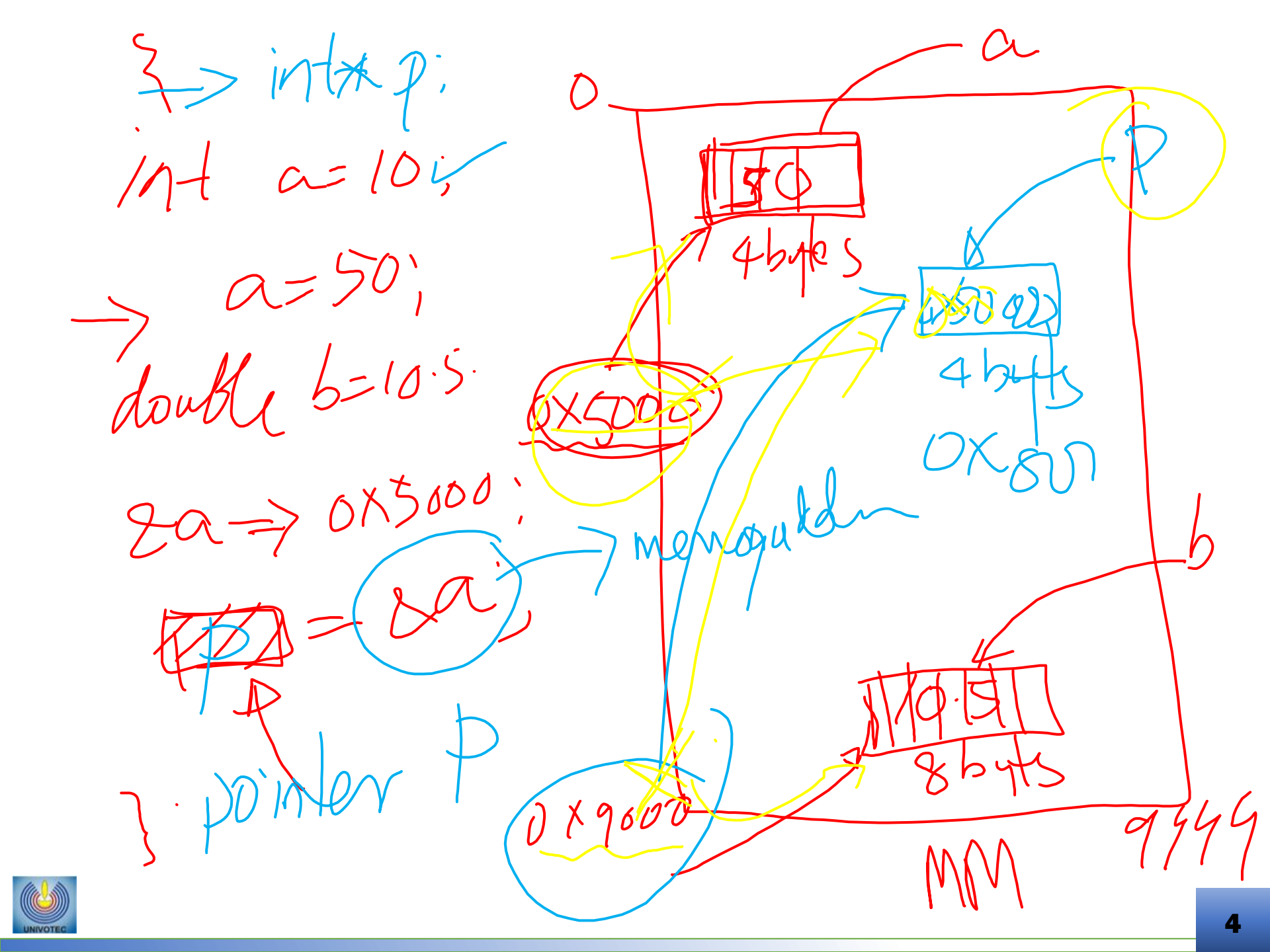
C++ Data Types (from old notes…)

- Data types in C++ is mainly divided into three types

  - ***Primitive Data Types:*** *These data types are built-in or predefined data types and can be used directly by the user to declare variables (int, double, char…)*

  - ***Derived Data Types:*** *The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types; (Array, Pointer, function, Reference)*

  - ***Abstract or User-Defined Data Types****: These data types are defined by user itself. (Classes, Structure, Enumeration, Typedef)*


int *p;
int a=10;
a=50;
double b=10.5;
&a => 0X5000;
p = &a;
pointer p
0
a
50
4bytes
p
0X5000
0X50
4bytes
memoria
b
10.5
8bytes
0X9000

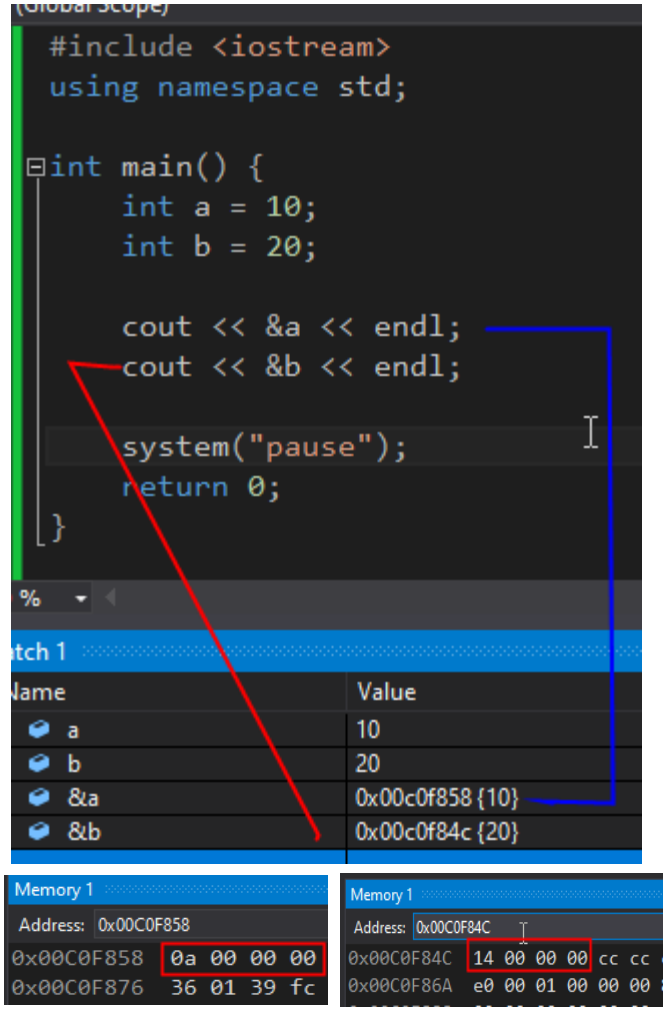
Derived Data Types (from old notes…)

- The data-types that are derived from the primitive are called **Derived Data Types**.

- These are four types
  - *References* ✓
  - *Arrays* ✓
  - *Function* ✓
  - *Pointers*

POINTER

- Every variable is occupied a memory location in the main memory,
- Every memory location has its own address
- By using ampersand (&) we can get those memory address
  - int a = 100;
  - “&a” will return the memory location (0x61fe1c)
- We called ampersand(&) as **reference operator** (address operator)



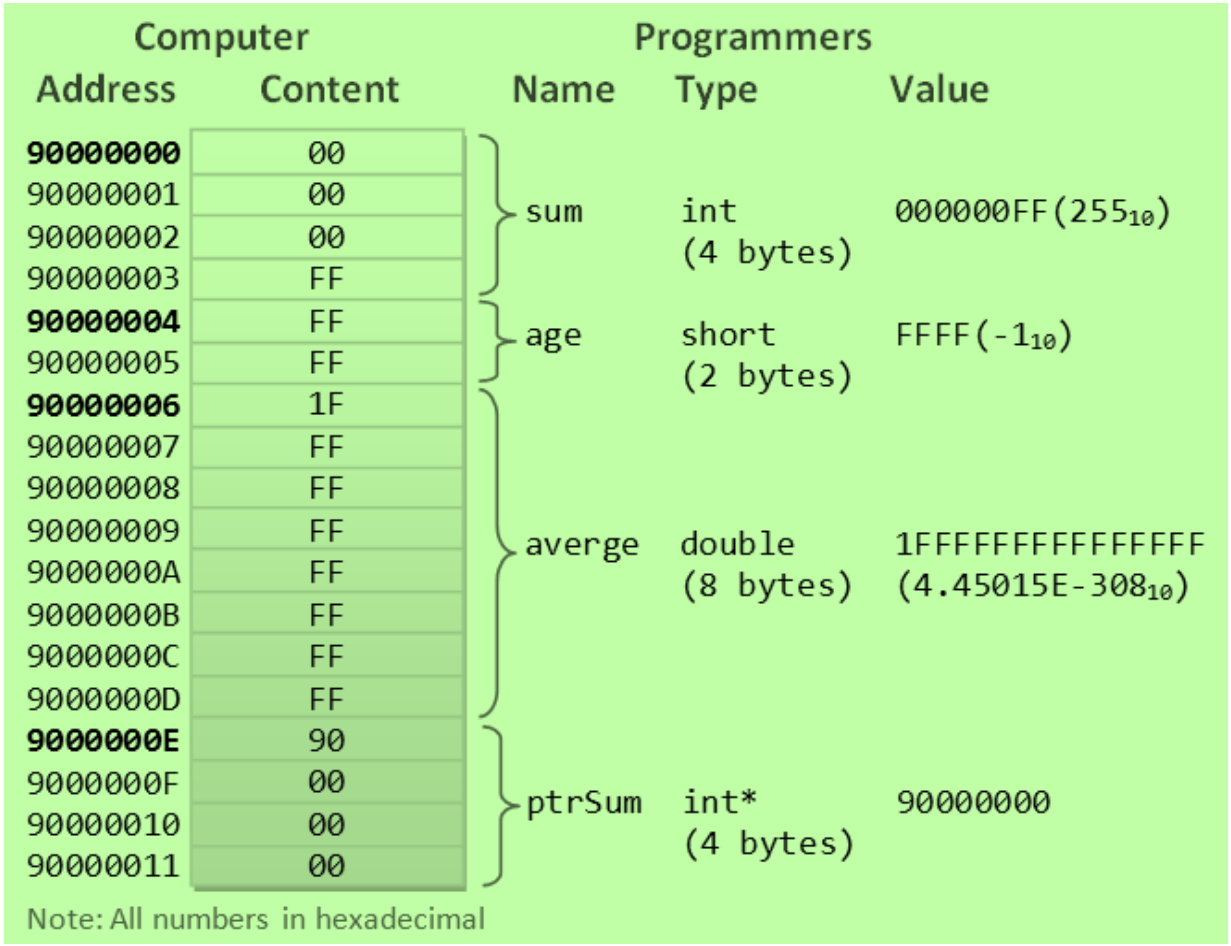

```
#include <iostream>
using namespace std;

int main() {
    int a = 100;
    cout << &a << endl;
    return 0;
}
```

```
0x61fe1c
```

RELATIONSHIPS

- Relationship between memory address and content, variable's name, type and value



Computer		Programmers		
Address	Content	Name	Type	Value
90000000	00			
90000001	00	sum	int (4 bytes)	000000FF(255_{10})
90000002	00			
90000003	FF			
90000004	FF	age	short (2 bytes)	FFFF(-1_{10})
90000005	FF			
90000006	1F	averge	double (8 bytes)	1FFFFFFFFFFFFFF ($4.45015E-308_{10}$)
90000007	FF			
90000008	FF			
90000009	FF			
9000000A	FF			
9000000B	FF			
9000000C	FF			
9000000D	FF			
9000000E	90	ptrSum	int* (4 bytes)	90000000
9000000F	00			
90000010	00			
90000011	00			

Note: All numbers in hexadecimal

Pointers are blocks of memory (4 bytes on 32-bit machines)

DECLARING A POINTER

- Now we need a special variable to store this memory address.
- So, we can use a pointer type variable to store a memory location
- Syntax:

```
<DataType> *<pointer_name>;
int *p;
double *num;
```

- Asterisk (*) sing is used to declare the pointer
- Now this pointer can hold any memory location in the memory which stored an integer variable

```
int a =10;
int *p;
p = &a;
```

INITIALIZING A POINTER

- You can initialize the pointer in creation or after you created
- Pointer p is initialized later once it has crated
- But pointer q is created & declared in same statement

```
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 20;

    int *p; //declare integer pointer
    p = &a; //Initialize

    int *q = &b; //declare & initialize

    system("pause");
    return 0;
}
```

%

tch 1

Name	Value
a	10
b	20
&a	0x00effa80 {10}
&b	0x00effa74 {20}
p	0x00effa80 {10}
q	0x00effa74 {20}

INITIALIZING A POINTER

- “p” is an integer type pointer, “a” & “b” are integer variables.
- First assign memory location where “a” was stored & then b’s memory location stored.
- Finally, it prints where pointer was located

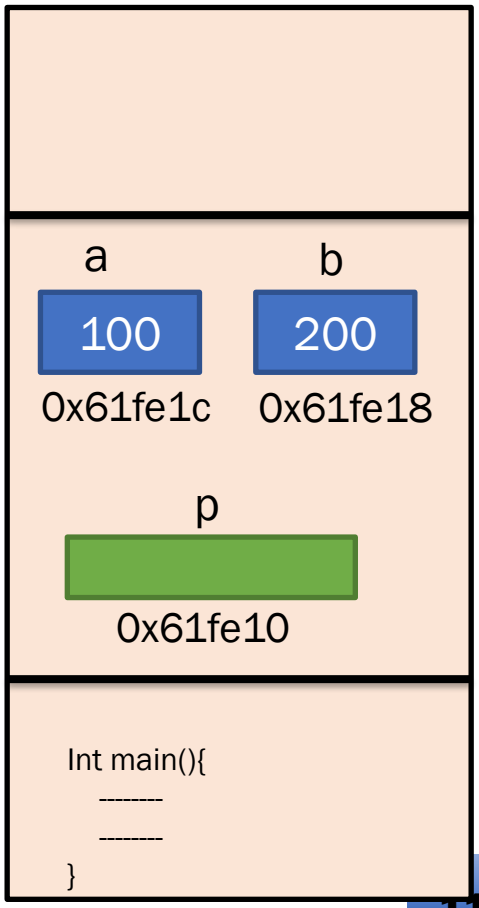
```
#include <iostream>
using namespace std;

int main() {
    int a = 100, b = 200;
    int *p;

    p = &a;
    cout << "&a = " << &a << " p = " << p << endl;
    p = &b;
    cout << "&b = " << &b << " p = " << p << " &p = " << &p <<  endl;

    return 0;
}
```

```
&a = 0x61fe1c p = 0x61fe1c
&b = 0x61fe18 p = 0x61fe18 &p = 0x61fe10
```



DEREFERENCE OPERATOR

- To get the value pointed by a pointer, we use the * operator.
- That is called dereference operator
- Value of “p” is a memory address where “a” variable is located
- Value of “*p” will give the value in this memory location (a’s memory location has value 100)

```
#include <iostream>
using namespace std;

int main() {
    int a = 100;
    int *p;
    p = &a;
    cout << "p = " << p << " *p = " << *p << endl;

    return 0;
}
```

```
p = 0x61fe14 *p = 100
```

CHANGE VALUE USING POINTER

- In below example a & b are integer variables
- p is the pointer to a
- *p can be used to read & write the value of “a”
- *p has used to assign value to b variable
- Also, *p used to change value of a

```
#include <iostream>
using namespace std;

int main() {
    int a = 100, b;
    int *p;     //declare a pointer
    p = &a;     //initializing pointer

    cout << "a  = " << a << endl;
    cout << "*p = " << *p << endl;

    b = *p; //Initialize b using pointer
    cout << "b  = " << b << endl;

    *p = 200;   //change a's value using pointer

    cout << "a  = " << a << endl;
    cout << "*p = " << *p << endl;
    cout << "b  = " << b << endl;

    system("pause");
    return 0;
}
```

```
a  = 100
*p = 100
b  = 100
a  = 200
*p = 200
b  = 100
```

POINTERS - TIPS

- Summery of pointer & variable

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 100;
    int* p = &i;

    cout << "i:      " << i << endl;
    cout << "p:      " << p << endl;
    cout << "&i:     " << &i << endl;
    cout << "&p:     " << &p << endl;
    cout << "*p:     " << *p << endl;
    cout << "*(&i): " << *(&i) << endl;

    return 0;
}
```

```
i:      100
p:      0073F868
&i:     0073F868
&p:     0073F85C
*p:     100
*(&i): 100
```

SIZE OF POINTERS

- Size of pointers are depending on the complier,
  - For 16-bit complier it is 2 bytes
  - For 32-bit complier it is 4 bytes
- But it is not change with data type

```cpp
#include <iostream>
using namespace std;

int main() {
    int i = 100;        int* ip = &i;
    double d = 120.5;   double* dp = &d;
    char c = 'A';       char* cp = &c;

    cout << "size of (ip): " << sizeof(ip) << endl;
    cout << "size of (dp): " << sizeof(dp) << endl;
    cout << "size of (cp): " << sizeof(cp) << endl;

    return 0;
}
```

```
size of (ip): 4
size of (dp): 4
size of (cp): 4
```

PASS BY ADDRESS (BY POINTER) – FROM OLD NOTES

- *You need to learn pointers to understand below example*

```
#include <iostream>
using namespace std;

void swap (int *x, int *y) {
     int temp;
     temp = *x;
     *x = *y;
     *y = temp;
}

int main() {
     int a, b;
     a = 10;
     b = 20;

     cout << "Before a = " << a << " b = " << b << endl;
     swap (&a, &b);
     cout << "After  a = " << a << " b = " << b << endl;

     return 0;
}
```

```
Before a = 10 b = 20
After  a = 20 b = 10
```

PASS BY ADDRESS (BY POINTER)

- In below it has passes address to swap() method

```
#include <iostream>
using namespace std;

void swap (int *x, int *y) {
     int temp;
     temp = *x;   //get 'a' value by dereferenceing
     *x = *y;     //get 'b' value by dereferenceing
     *y = temp;
}

int main() {
     int a =10, b = 20;

     cout << "Before a = " << a << " b = " << b << endl;
     swap (&a, &b);  //pass address
     cout << "After  a = " << a << " b = " << b << endl<<endl;


     int *p = &a;
     int *q = &b;
     cout << "Before a = " << a << " b = " << b << endl;
     swap (p, q);     //pass pointers
     cout << "After  a = " << a << " b = " << b << endl;

     return 0;
}
```

```
Before a = 10 b = 20
After  a = 20 b = 10

Before a = 20 b = 10
After  a = 10 b = 20
```

NULL POINTER

- We can initialize a pointer to 0 or NULL, which is pointing to nothing
- It is always a good practice to assign the pointer NULL if you are not certain the address
- A pointer that is assigned NULL is called a **null pointer**.
- The NULL pointer is a constant with a value of zero

```
int   *ptr = NULL;
```

POINTER OPERATIONS

- Some arithmetic operators can be used with pointers:
- Increment and decrement operators (++, --)
- Each time a pointer is incremented by 1, it points to the memory location of the next element of its base type.
  - If “p” is a character pointer then “p++” will increment “p” by 1 byte.
  - If “p” were an integer pointer its value on “p++” would be incremented by 4 bytes
  - If “p” were an integer pointer its value on “p--” would be decrement by 4 bytes
- If “p” were an integer pointer, then value of (p+1) is same as p++.
  - But p++ will change memory address after the execution

POINTER OPERATIONS

- Arithmetic operations

```
#include <iostream>
using namespace std;

int main() {
    int a = 100;
    int *ip = &a;

    cout << "ip   = " << ip << endl << endl;
    for (int i = -5; i <= 5; i++){
        cout << "ip" << std::showpos<<i
            << " = " << ip + i << endl;
    }


    system("pause");
    return 0;
}
```

```
ip    = 012FFDD0

ip-5 = 012FFDBC
ip-4 = 012FFDC0
ip-3 = 012FFDC4
ip-2 = 012FFDC8
ip-1 = 012FFDCC
ip+0 = 012FFDD0
ip+1 = 012FFDD4
ip+2 = 012FFDD8
ip+3 = 012FFDDC
ip+4 = 012FFDE0
ip+5 = 012FFDE4
```

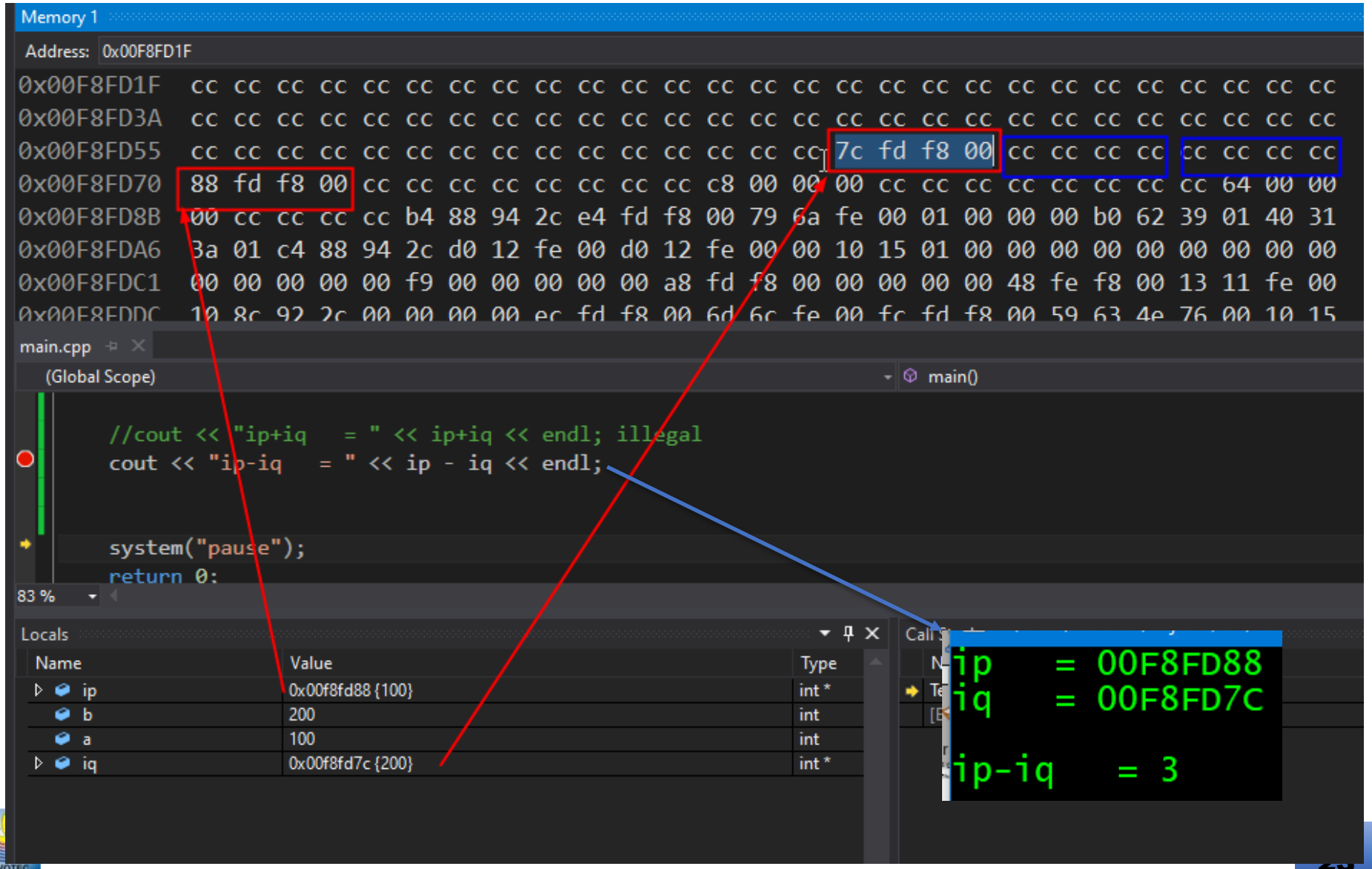
```
double a = 100;
double *ip = &a;

cout << "ip   = " << ip << endl << endl;
for (int i = -5; i <= 5; i++){
    cout << "ip" << std::showpos<<i
        << " = " << ip + i << endl;
}
```

```
ip    = 00F9F8C0

ip-5 = 00F9F898
ip-4 = 00F9F8A0
ip-3 = 00F9F8A8
ip-2 = 00F9F8B0
ip-1 = 00F9F8B8
ip+0 = 00F9F8C0
ip+1 = 00F9F8C8
ip+2 = 00F9F8D0
ip+3 = 00F9F8D8
ip+4 = 00F9F8E0
ip+5 = 00F9F8E8
```

POINTER OPERATIONS



POINTER OPERATIONS

- valid operations
- p1 – p2
- p++
- p--

- p1 + p2 is an invalid operations, since out or memory range
- (*p1 + *p2) is valid, since it is working with values

```
#include <iostream>
using namespace std;

int main() {
    int a =10;;
    int *p = &a;

    cout << "Original p = " << p << endl;
    p++;
    cout << "After p++ = " << p << endl;

    cout << "Value of p + 1 = " << (p + 1)  << endl;
    cout << "Value of p + 2 = " << (p + 2)  << endl;
    cout << "Value of p + 3 = " << (p + 3)  << endl;
    cout << "Value of p - 1 = " << (p - 1)  << endl;

    return 0;
}
```

```
Original p = 0x61fe14
After p++ = 0x61fe18
Value of p + 1 = 0x61fe1c
Value of p + 2 = 0x61fe20
Value of p + 3 = 0x61fe24
Value of p - 1 = 0x61fe14
```

POINTER OPERATIONS

```
int main() {
    int a = 100, b=200;
    int *ip = &a, *iq = &b;

    cout << "ip    = " << ip << endl;
    cout << "iq    = " << iq << endl;
    cout << "*ip   = " << *ip << endl;
    cout << "*iq   = " << *iq << endl << endl;

    //cout << "ip + iq    = " << ip+iq << endl; //illegal
    //cout << "ip * iq    = " << ip*iq << endl; //illegal
    //cout << "ip / iq    = " << ip/iq << endl; //illegal
    //cout << "ip % iq    = " << ip%iq << endl; //illegal
    cout << "ip - iq      = " << ip - iq << endl;
    cout << "++ip         = " << ++ip << endl;
    cout << "--ip         = " << --ip << endl;

    cout << "*ip + *iq    = " << *ip + *iq << endl;
    cout << "*ip - *iq    = " << *ip - *iq << endl;
    cout << "*ip * *iq    = " << *ip * *iq << endl;
    cout << "*iq / *ip    = " << *iq/(*ip) << endl;
    cout << "*ip % *iq    = " << *ip % *iq << endl;

    system("pause");
    return 0;
}
```

```
ip    = 0133FB10
iq    = 0133FB04
*ip   = 100
*iq   = 200

ip - iq      = 3
++ip         = 0133FB14
--ip         = 0133FB10
*ip + *iq    = 300
*ip - *iq    = -100
*ip * *iq    = 20000
*iq / *ip    = 2
*ip % *iq    = 100
Press any key to continue . . .
```

POINTERS AND ARRAYS

- Let's print memory locations of an array.
- Assign pointer to array & print again using pointer

```
#include <iostream>
using namespace std;

int main() {
    int arr [5] = {10,20,30,40,50};
    cout << " Memory locations of the array" << endl;
    for (int i=0; i<5; i++) {
        cout << " &arr["<<i<<"] = " << &arr[i] << endl;
    }

    int *p = arr;
    cout << endl << " Using pointer" << endl;
    for (int i=0; i<5; i++) {
        cout << " p + "<<i<<" = " << p++ << endl;
    }

    return 0;
}
```

```
Memory locations of the array
&arr[0] = 0x61fdf0
&arr[1] = 0x61fdf4
&arr[2] = 0x61fdf8
&arr[3] = 0x61fdfc
&arr[4] = 0x61fe00

Using pointer
p + 0 = 0x61fdf0
p + 1 = 0x61fdf4
p + 2 = 0x61fdf8
p + 3 = 0x61fdfc
p + 4 = 0x61fe00
```

POINTERS AND ARRAYS

- Array name itself is a pointer to first element of the array

```cpp
#include <iostream>
using namespace std;

int main() {
    int arr [5] = {10,20,30,40,50};
    cout << " Memory locations of the array" << endl;
    for (int i=0; i<5; i++) {
        cout << " &arr["<<i<<"] = " << &arr[i] << endl;
    }

    int *p = arr;

    cout << endl << " arr = " << arr << endl;
    for (int i=0; i<5; i++) {
        cout << " arr + "<<i<<" = " << arr + i  << endl;
    }

    cout << endl << " *arr = " << *arr << endl;
    for (int i=0; i<5; i++) {
        cout << " *(arr + "<<i<<") = " << *(arr + i)  << endl;
    }
    cout << endl << " *p = " << *p << endl;
    for (int i=0; i<5; i++) {
        cout << " *(p + "<<i<<") = " << *(p + i)  << endl;
    }

    return 0;
}
```

```
Memory locations of the array
&arr[0] = 0x61fdf0
&arr[1] = 0x61fdf4
&arr[2] = 0x61fdf8
&arr[3] = 0x61fdfc
&arr[4] = 0x61fe00

arr = 0x61fdf0
arr + 0 = 0x61fdf0
arr + 1 = 0x61fdf4
arr + 2 = 0x61fdf8
arr + 3 = 0x61fdfc
arr + 4 = 0x61fe00

*arr = 10
*(arr + 0) = 10
*(arr + 1) = 20
*(arr + 2) = 30
*(arr + 3) = 40
*(arr + 4) = 50

*p = 10
*(p + 0) = 10
*(p + 1) = 20
*(p + 2) = 30
*(p + 3) = 40
*(p + 4) = 50
```

POINTER VS REFERENCE

- A pointer can be re-assigned while reference cannot and must be assigned at initialization only.
- Pointer can be assigned NULL directly, whereas reference cannot.
- Pointers can use ++ to go to the next item that a pointer is pointing to (we used in arrays)
- A pointer is a variable that holds a memory address. A reference has the same memory address as the item it references.
- A pointer needs to be dereferenced with * to access the memory location it points to, whereas a reference can be used directly.

POINTER TO POINTER

- A pointer to a pointer is a form of multiple indirection or a chain of pointers.
- Normally, a pointer contains the address of a variable.
- When we define a pointer to a pointer, the first pointer contains the address of the second pointer,
- which points to the location that contains the actual value as shown below.
- Syntax:

```
<data_type> ** <variable_name>;
int  **pptr;
```

POINTER TO POINTER

- When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example –

```
#include <iostream>
using namespace std;

int main() {
    int  var = 100;
    int  *ptr;
    int  **pptr;

    ptr = &var;
    pptr = &ptr;

    cout << "Value of var :" << var << endl;
    cout << "Value available at *ptr :" << *ptr << endl;
    cout << "Value available at **pptr :" << **pptr << endl;

    return 0;
}
```

```
Value of var :100
Value available at *ptr :100
Value available at **pptr :100
```

UoVT

UNIVERSITY OF VOCATIONAL TECHNOLOGY

Phone: +94-112630700
Fax: +94-112630705
Email: univotec@univotec.ac.lk
Web: http://univotec.ac.lk/