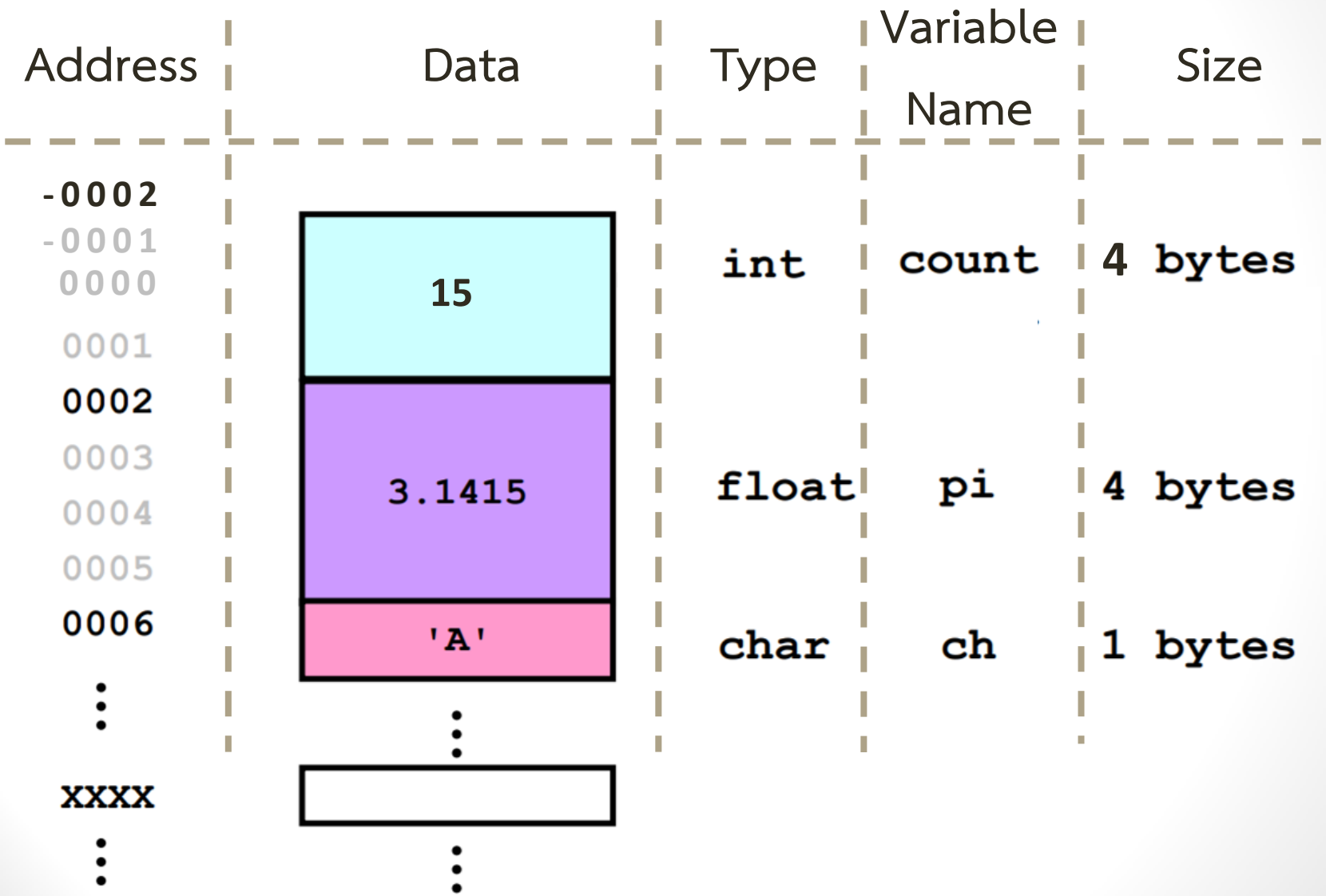
Chapter 8

Pointers 1

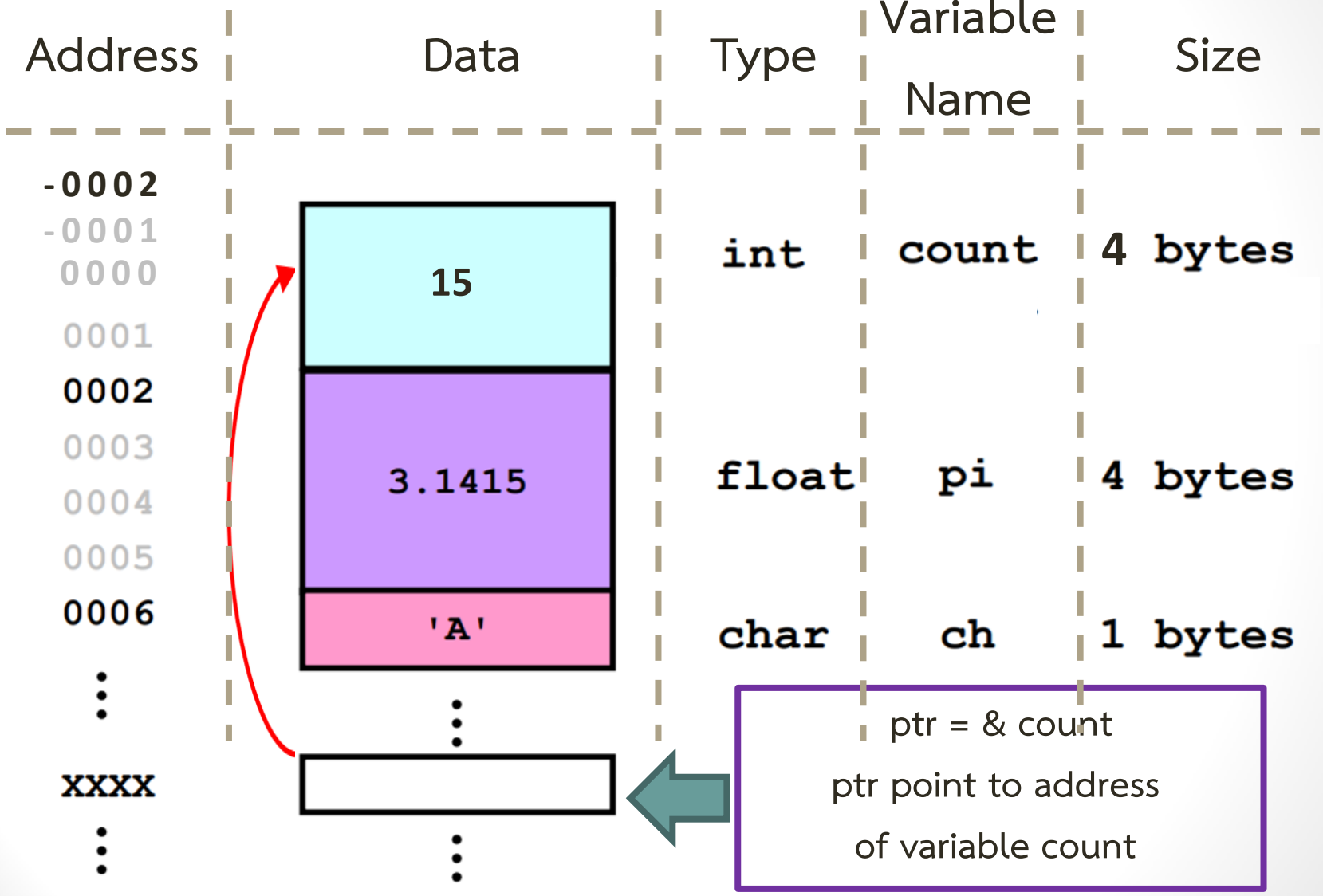
Faculty of Engineering KMITL
King Mongkut's Institute of Technology Ladkrabang

Memory and Pointer Structure



Reference..www.ce.kmitl.ac.th King Mongkut's Institute of Technology Ladkrabang

Memory and Pointer Structure



Reference..www.ce.kmitl.ac.th King Mongkut's Institute of Technology Ladkrabang

Pointer Declaration

```
type        *pointer_name;
```

type is type of pointer variable

***** is operator that identify pointer

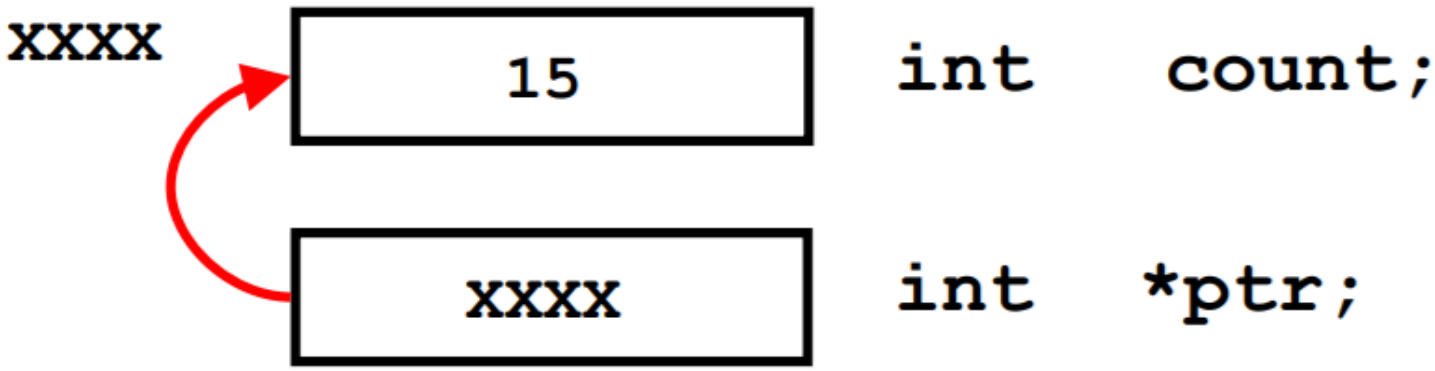
pointer_name is name of pointer variable

<u>Example</u>

```
int     * ptr_int;      /* pointer to integer */
float   * ptr_float;    /* pointer to float */
char    * ptr_char;     /* pointer to char */
```

Reference Operator “&”

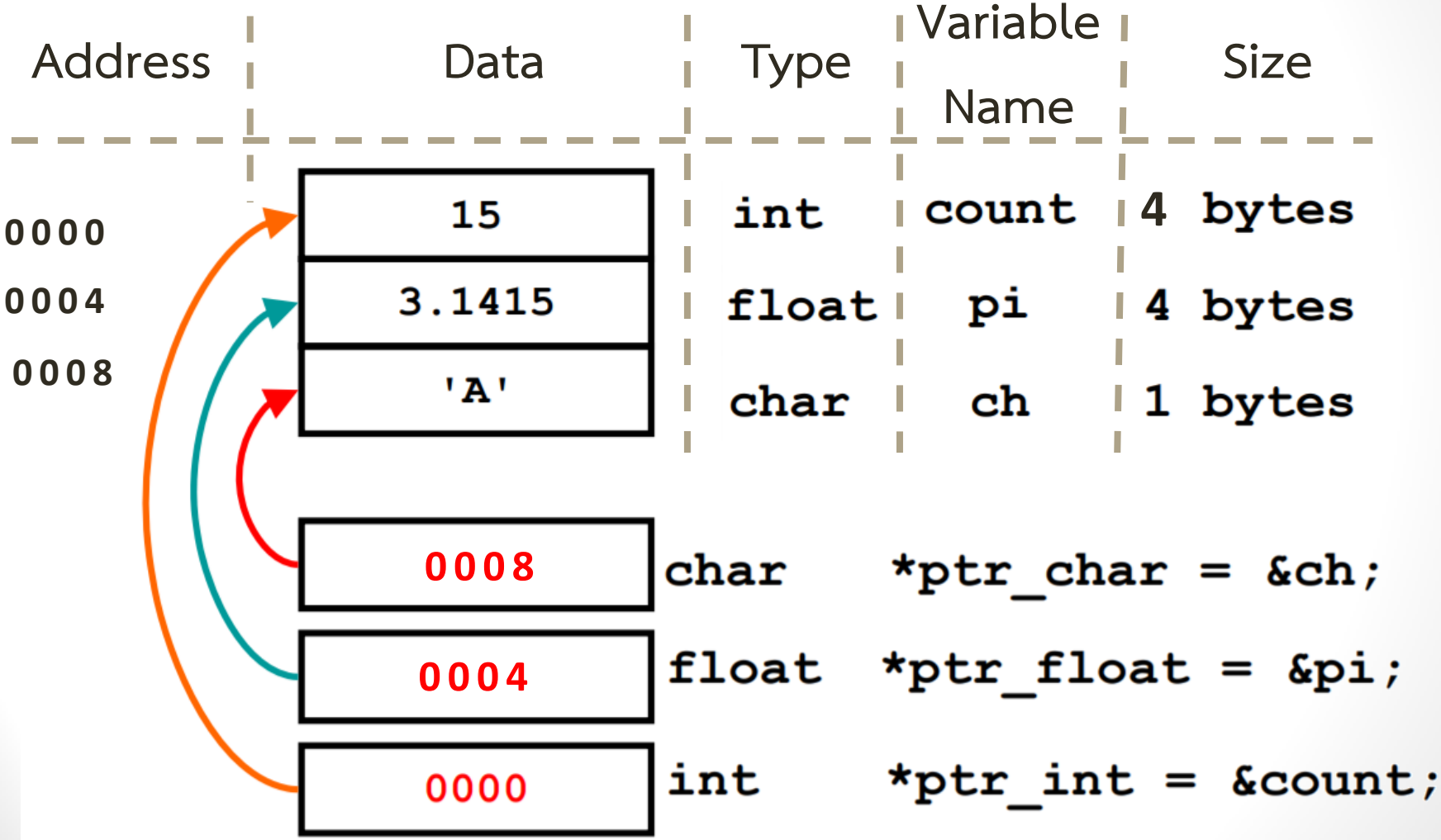
Reference Operator gives you the address of a variable.




```
int    count;
int    *ptr;
```

```
ptr = &count;          /* ptr is XXXX */
```

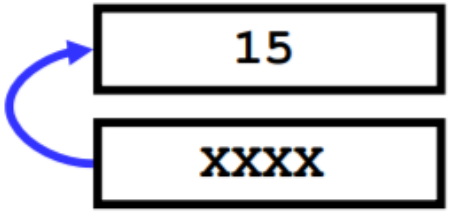
Reference Operator “&”



Indirect Operator "*"

Indirection or Dereferencing gets you the value from the address

Address XXXX




```
int   count;
int   * ptr;
```

```
int     count = 15, y, z[10];
int    *ptr;          /* ptr is int pointer   */

ptr   = &count;       /* ptr point to count */
y     = *ptr;         /*  y = 15              */
*ptr  =  0;           /*  count = 0           */
ptr   = &z[0];        /* ptr point to z [0]   */
```

Shows pointer by printf function

Function printf can show address by using this operator.

- %p show address in hexadecimal
- %u show address in decimal format

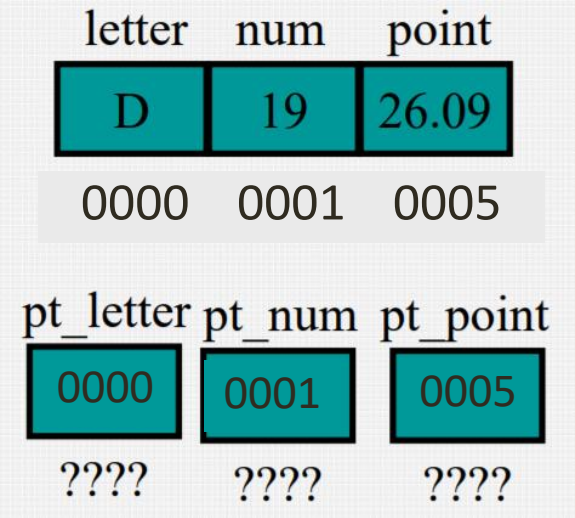
The result will show in format XXXX:YYYY or XXXX depend on memory model

```
1  #include <stdio.h>
2  int main()
3  {
4      int i = 10;
5      int *p;
6      p = &i;
7      printf("%p  %d\n",&i, i);
8      printf("%p  %p  %d\n",&p, p, *p);
9      printf("--------------\n");
10     printf("%u  %d\n",&i, i);
11     printf("%u  %u  %d\n",&p, p, *p);
12     return 0;
13 }
```

```
C:\WINDOWS\system32\cmd.exe
0012FF60  10
0012FF54  0012FF60  10
--------------
1245024  10
1245012  1245024  10
Press any key to continue .
```

Example : Program shows address by pointer

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char    letter = 'D';
    int     num = 19;
    float   point = 26.09;
    char    *pt_letter;
    int     *pt_num;
    float   *pt_point;
    pt_letter  =   &letter;
    pt_num     =   &num;
    pt_point   =   &point;
    printf("Address of letter = %p \n",pt_letter);
    printf("Address of num = %p \n",pt_num);
    printf("Address of point = %p \n",pt_point);
  return 0;
}
```

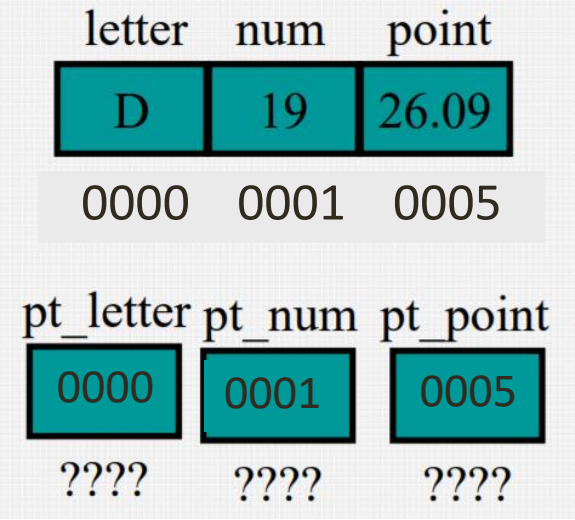


Example : Program shows address by pointer

Address of letter = 0000

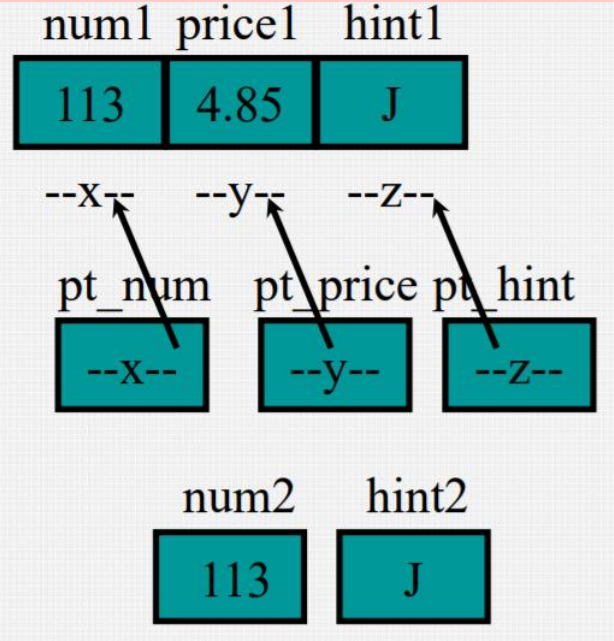
Address of num = 0002

Address of point = 0004



Example : Program shows referencing and dereferencing

```
int main()
{
        int    num1 = 113,num2;
        float  price1 = 4.85;
        char   hint1 = 'J',hint2;
        int    *pt_num;
        float  *pt_price;
        char   *pt_hint;
        pt_num    =  &num1;
        pt_price  =  &price1;
        pt_hint   =  &hint1;
        num2      =  *pt_num;
        hint2     =  *pt_hint;
        printf ("Variable num1 = %d \n", num2);
        printf ("Variable price1 = %f \n",*pt_price);
        printf ("Variable hint2 = %c \n", hint2);
    return 0;
}
```

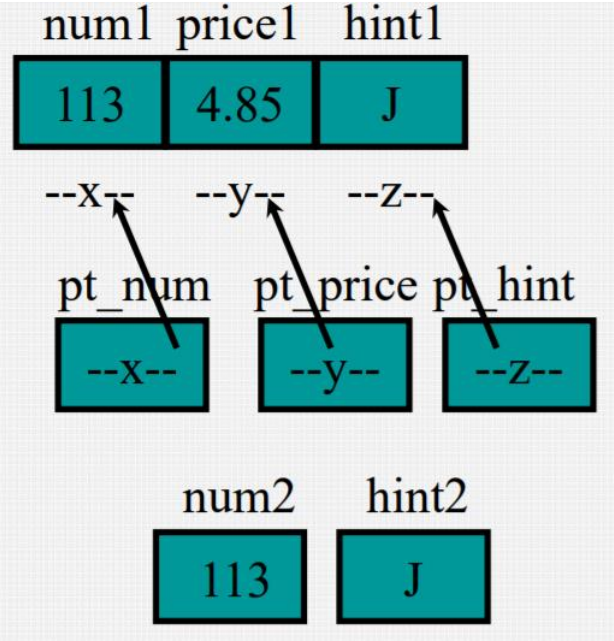


Example : Program shows address by pointer

Variable num1 = 113

Variable price1 = 4.850000

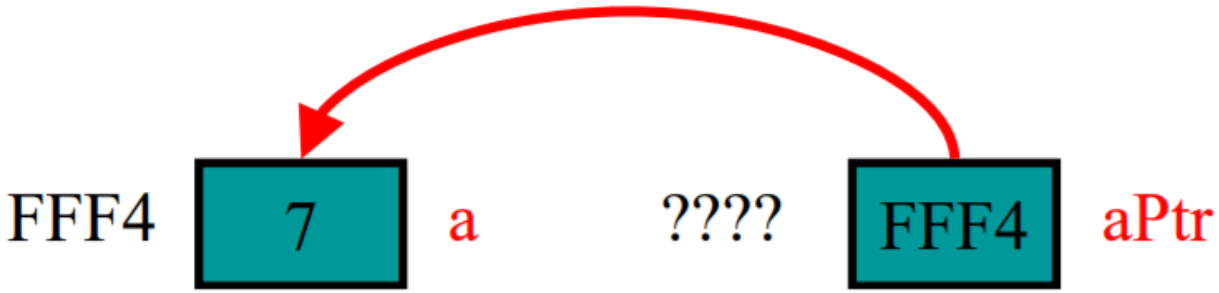
Variable hint2 = J



Example : Program shows referencing and dereferencing 2

```
#include<stdio.h>
#include<conio.h>
int main()
{
        int a;              /* a is an integer */
        int *aPtr;          /* aPtr is a pointer to an integer */
        a = 7;
        aPtr = &a;          /* aPtr set to address of a */
printf ("The address of a is %p\n"
         "The value of aPtr is %p\n\n", &a, aPtr);
printf ("The value of a is %d\n"
         "The value of *aPtr is %d\n\n", a, *aPtr);
printf ("Proving that * and & are complements of "
         "each other.\n&*aPtr = %p\n*&aPtr = %p\n",
         &*aPtr, *&aPtr);
return 0;
}
```

Example : Program shows referencing and dereferencing 2



The address of a is FFF4

The value of aPtr is FFF4

The value of a is 7

The value of *aPtr is 7

Proving that * and & are complements of each other.

&*aPtr = FFF4

*&aPtr = FFF4

Example : Pointer Conclusion

Pointer Declaration

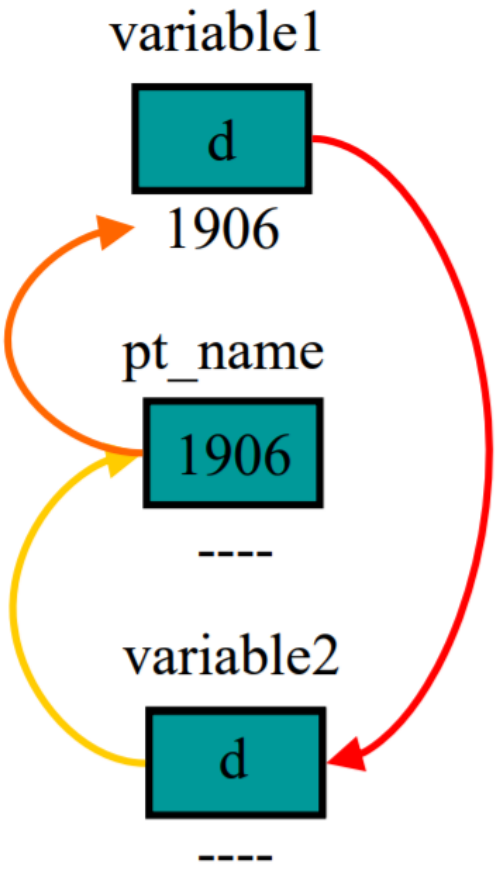
```
type          *pointer_name;
```

Show address by using “&”

```
pt_name  =  &variable1;
```

Show value by using “*”

```
variable2  =  *pt_name;
```



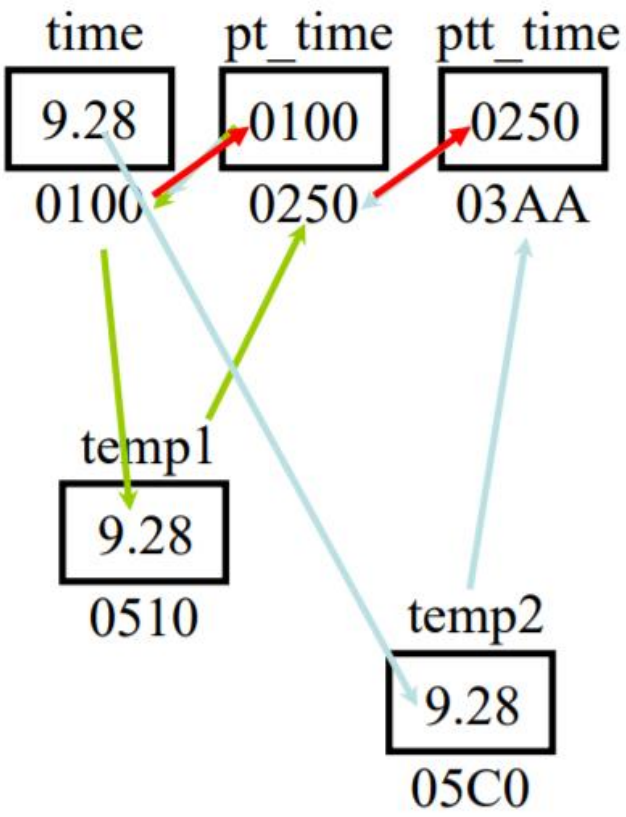
Pointer to Pointer

```
type        **ptt_name;
```

type	is type of pointer variable
****	is operator that identify pointer to pointer
ptt_name	is name of pointer to pointer variable

Pointer to Pointer

```
float time = 9.28;
float *pt_time;
float **ptt_time;
pt_time = &time;
ptt_time = &pt_time;
float temp1;
temp1 = *pt_time;
float temp2;
temp2 = **ptt_time;
```



Pointer and Array

Arrays are closely related to pointers in C programming but the important difference between them is that, a pointer variable takes different addresses as value where as, in case of array it is fixed.

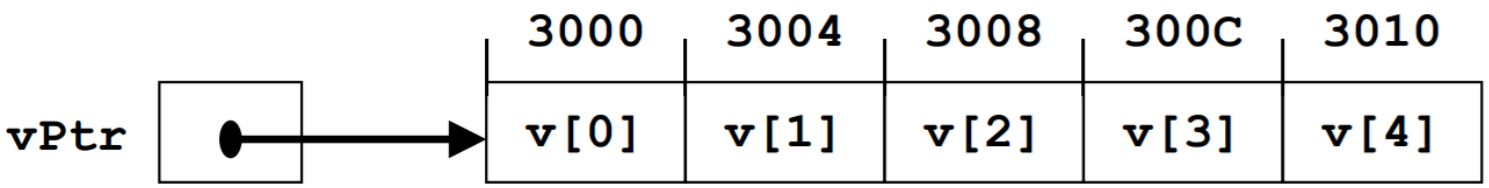
<u>Example</u>

float v[5]

Definition Array **v** size **5** which mean group of continuous object. Name are **v[0], v[1], v[2], v[3], v[4].**

3000	3004	3008	300C	3010
v[0]	v[1]	v[2]	v[3]	v[4]

Pointer and Array



Declaration **float *vPtr** and define **vPtr** point to Array address **v** can do by 2 methods.

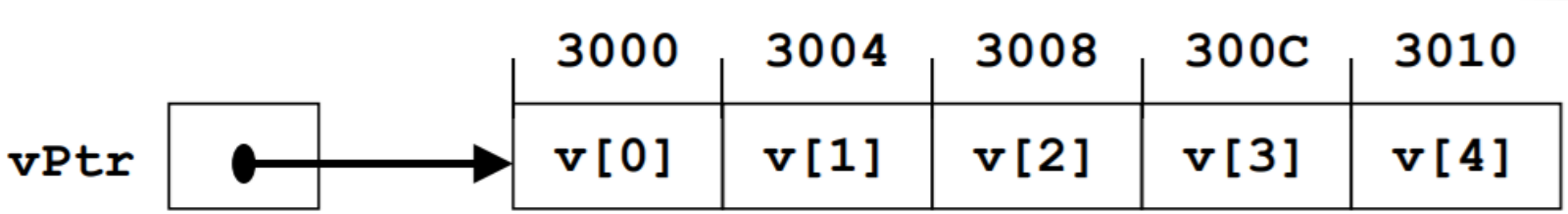
<u>Method 1</u>

vPtr = v;

<u>Method 2</u>

vPtr = &v[0];

Pointer and Array



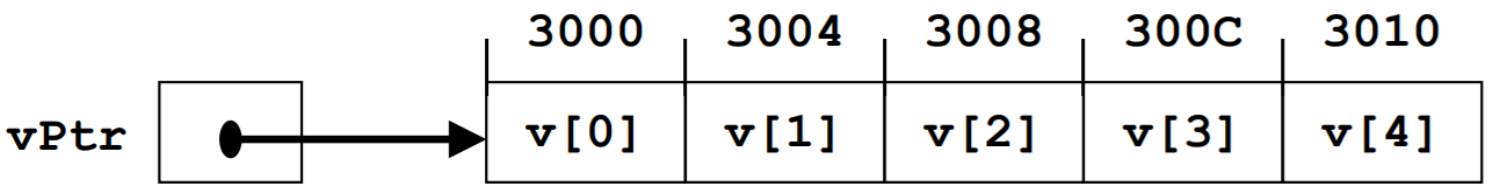
Meaning of definition **x = *vPtr** is copy value of **v[0]** to **x**

Move Array Pointer by += and -=

Pointer can be used as Arithmetic Operator as below.

- Increment (++) or Decrement (--)
- Integer can add or minus with pointer (+, +=) or (-, -=)
- Pointer can minus with another Pointer.

Move Array Pointer by += and -=



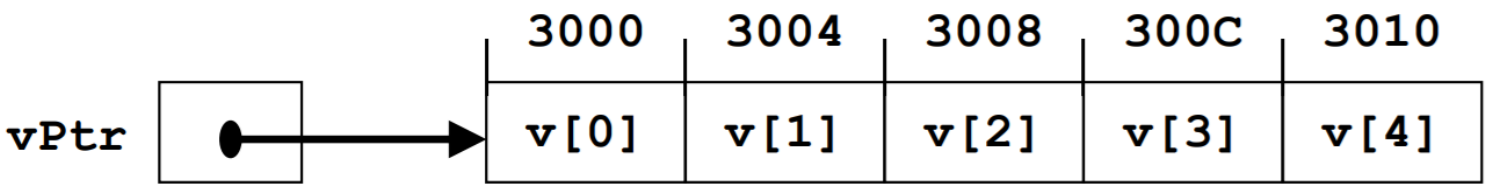
When add or minus Pointer and integer then

- Value of pointer not increase or decrease by that number.
- Value of pointer will increase or decrease by that number multiply with size of object pointed.
- Size (byte) depend on type of object.

<u>Example</u> (Define size of float object is 4 byte)

```
vPtr += 2;        //        vPtr = vPtr+(2*4)
                  //   or   vPtr = 3000+(2*4)
```

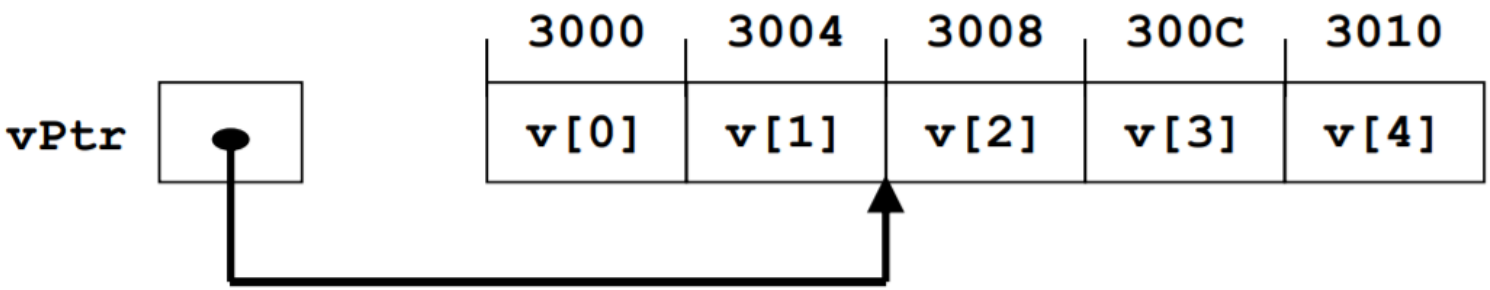
Move Array Pointer by += and -=



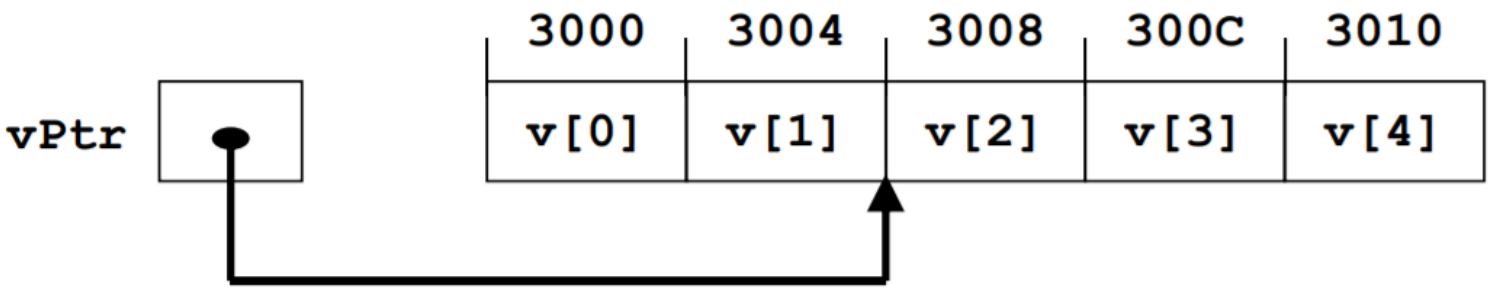
vPtr += 2; // vPtr = vPtr+(2*4)

// or vPtr = 3000+(2*4)

After above method **vPtr** will point to **v[2]**



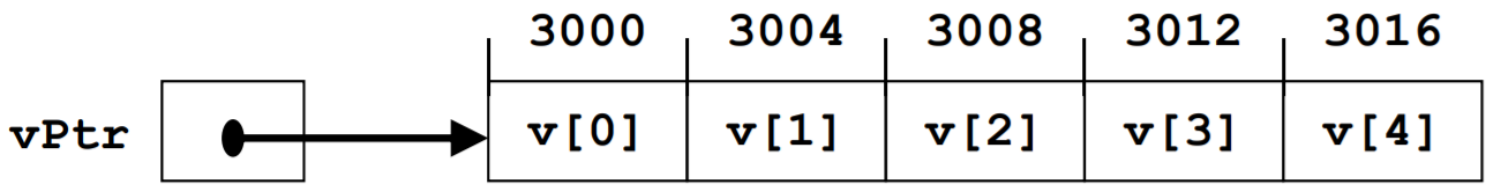
Move Array Pointer by += and -=



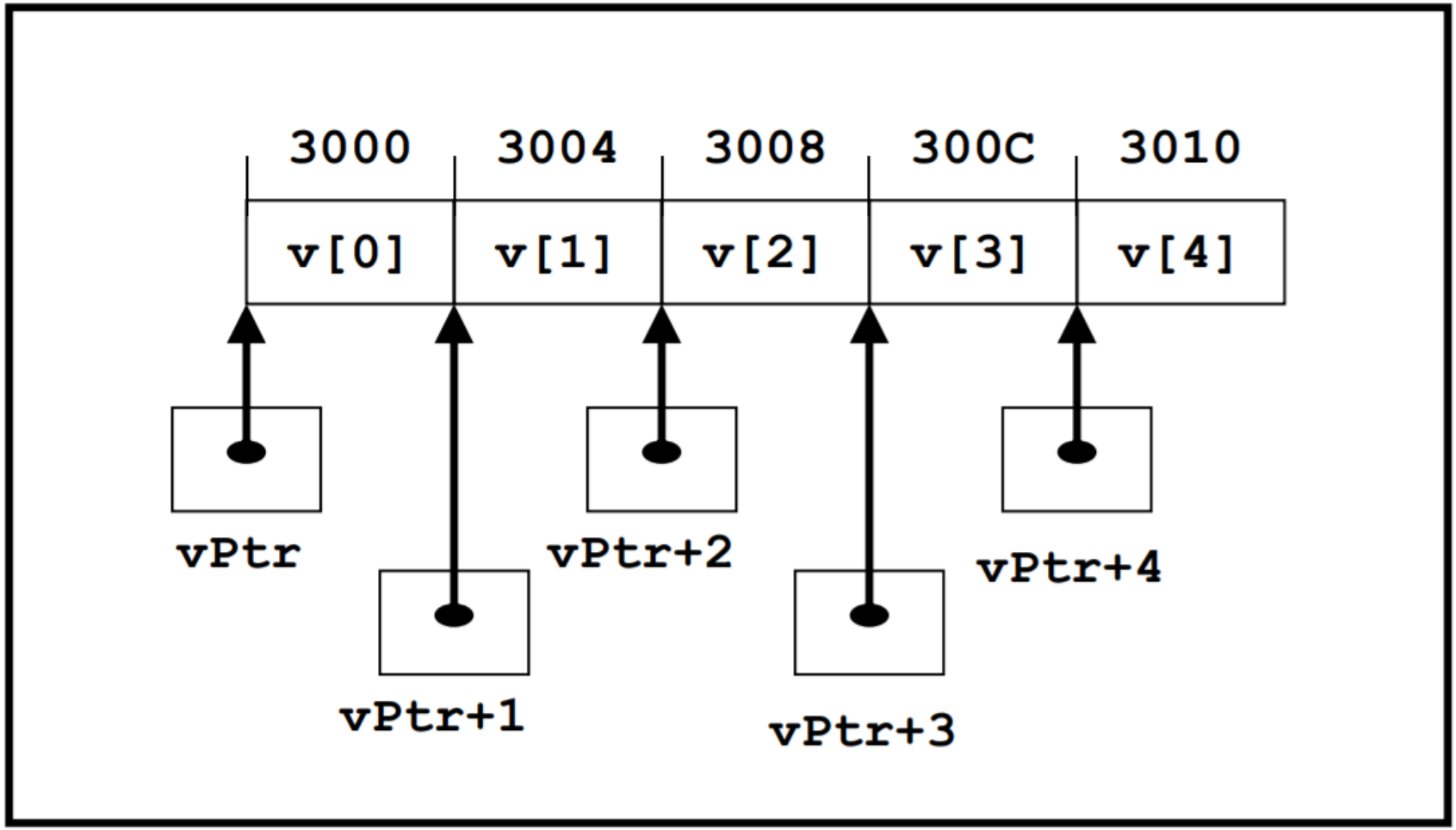
vPtr -= 2; // vPtr = vPtr-(2*4)

// or vPtr = 3008-(2*4)

After above method **vPtr** will point to **v[0]**

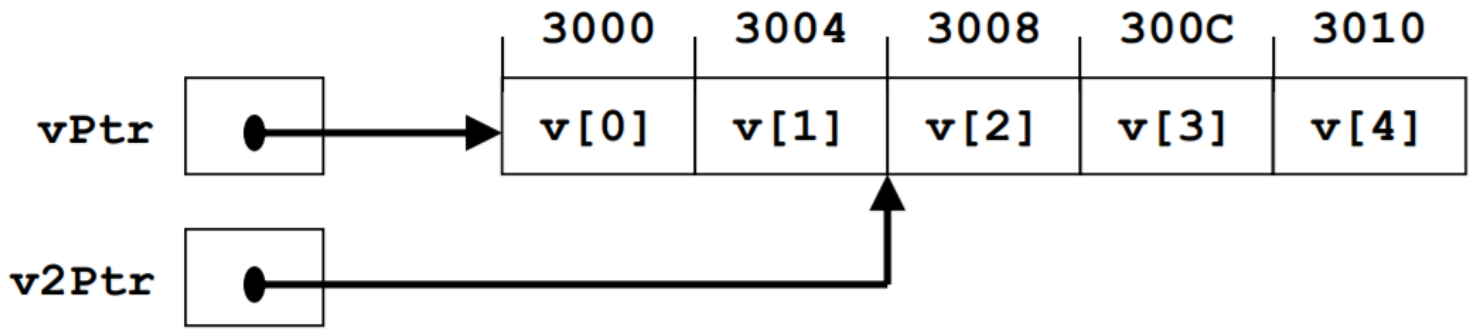


Move Array Pointer by += and -=



Reference..www.ce.kmitl.ac.th King Mongkut's Institute of Technology Ladkrabang

Calculate number of element by Pointer

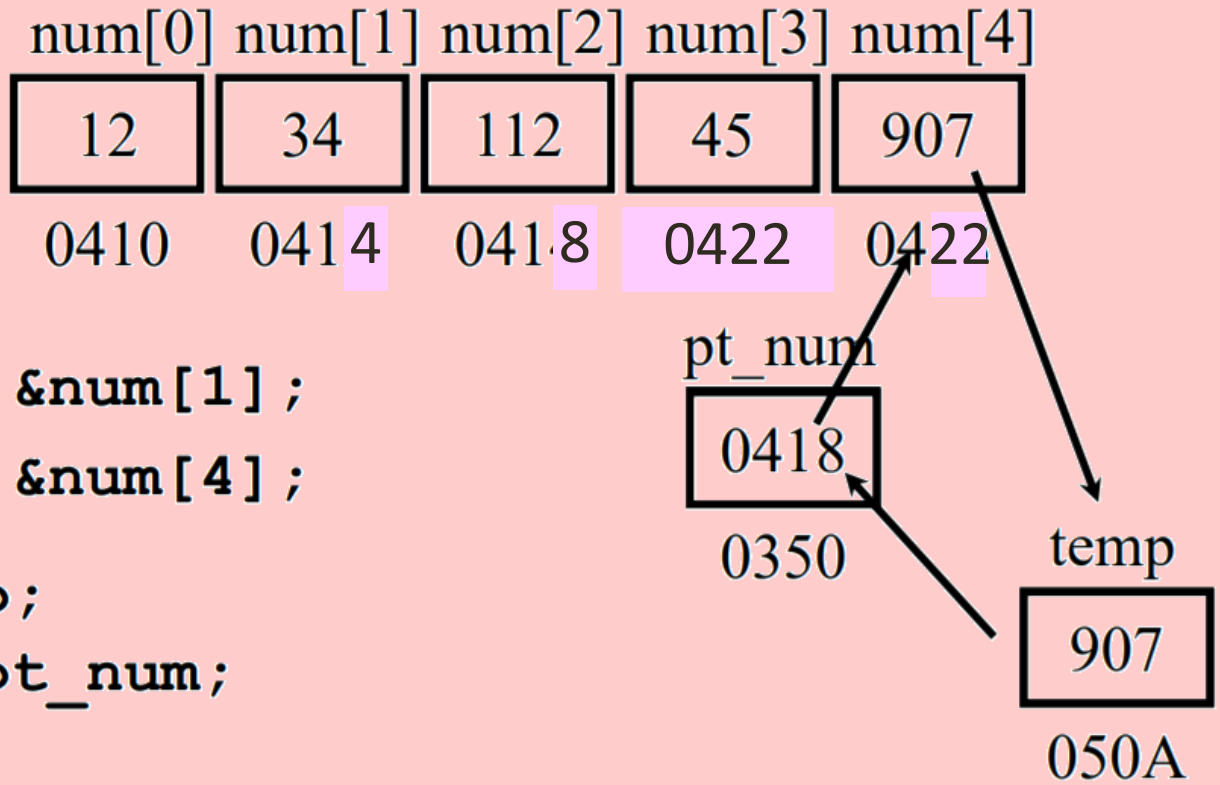



```
• vPtr = &v[0];          //      vPtr = 3000
• v2Ptr = &v[2];         // or v2Ptr = 3008
• x = v2Ptr - vPtr;      // x = ?
```

Value of **x** is number of element of Array variable from **vPtr** to **vPtr2** in this case is **2**

Example: Program show how to use pointer and array

```
int  num[5] = {12, 34, 112, 45, 907};
int  *pt_num;
```




```
pt_num = &num[1];
pt_num = &num[4];

int  temp;
temp = *pt_num;
```

Example: Program show how to use pointer and array

```
type name[10];
type *pt_name;
pt_name = name;      //pt_name = &name[0]
-----------------------------------------

 float num[] = {19.01, 26.09, -4.23, -4.24, -13.12}
```

num[0]	num[1]	num[2]	num[3]	num[4]
19.01	26.09	-4.23	-4.24	-13.12
0300	0304	0308	030C	0310

Example: Program show how to use pointer and array

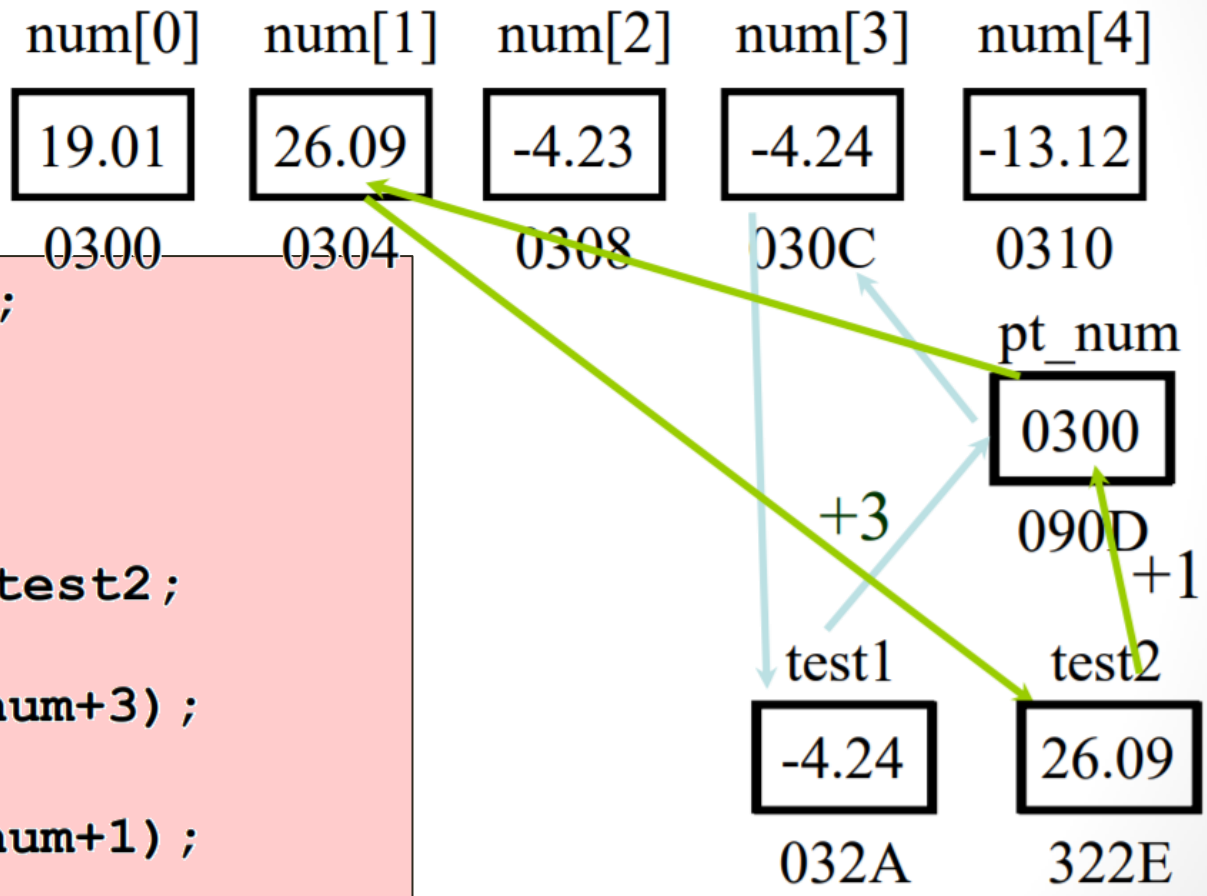
```
float  *pt_num;

pt_num = num;

float  test1, test2;

test1 = *(pt_num+3);

test2 = *(pt_num+1);
```



Relation between Array and Pointers

Arrays are closely related to pointers which almost use in every case.

```
int b[5];
int *bPtr;
bPtr = b;          //equivalent to   bPtr = &b[0];
&b[3] equivalent to bPtr+3
b[3] equivalent to *(bPtr+3)
```

Case 1 : Normal Array

```
int main()
{
        int i, offset, b[] = {10, 20, 30, 40};
        int *bPtr = b;        /* set bPtr to point to array b */
        printf("Array b printed with:\n"
                "Array subscript notation\n");
        for (i=0; i<=3; i++)
                printf("b[%d] = %d\n", i, b[i]);
        return 0;
}
```

```
Array b printed with:
Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40
```

Case 2 : Array in Pointer format

```
int main()
{
        int i, offset, b[] = {10, 20, 30, 40};
        int *bPtr = b;              /* set bPtr to point to array b */
        printf("Pointer/offset notation where\n"
                "the pointer is the array name\n");
        for (offset = 0; offset<=3 ; offset++)
                printf("*(b + %d) = %d\n", offset, *(b + offset));
        return 0;
}
```

Pointer/offset notation where

The pointer is the array name

*(b + 0) = 10

*(b + 1) = 20

*(b + 2) = 30

*(b + 3) = 40

Case 3 : Pointer in Array format

```
int main()
{
    int i, offset, b[] = {10, 20, 30, 40};
    int *bPtr = b;        /* set bPtr to point to array b */
    printf("Pointer subscript notation\n");
    for (i=0 ; i<=3 ; i++)
        printf("bPtr[%d] = %d\n", i, bPtr[i]);
    return 0;
}
```

Pointer subscript notation

bPtr[0] = 10

bPtr[1] = 20

bPtr[2] = 30

bPtr[3] = 40

Case 4 : Normal Pointer

```
int main()
{
    int i, offset, b[] = {10, 20, 30, 40};
    int *bPtr = b;     /* set bPtr to point to array b */
    printf("Pointer/offset notation\n");
    for (offset=0; offset<=3; offset++)
        printf("*(bPtr + %d) = %d\n", offset, *(bPtr + offset));
    return 0;
}
```

Pointer/offset notation

*(bPtr + 0) = 10

*(bPtr + 1) = 20

*(bPtr + 2) = 30

*(bPtr + 3) = 40