

ED1021 - Introduction to computation and visualisation

L12 - Pointers and Arrays in C

Ramanathan Muthuganapathy (<https://ed.iitm.ac.in/~raman>)

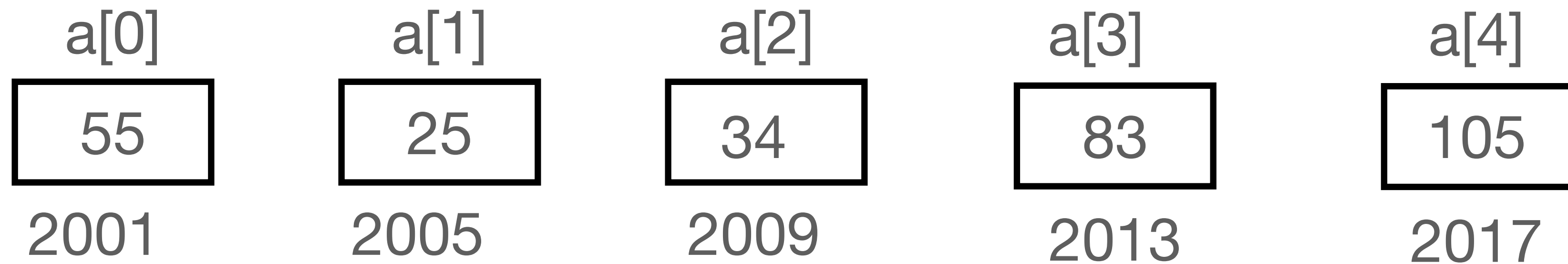
Course web page: <https://ed.iitm.ac.in/~raman/introcomp.html>

Moodle page: Available at <https://courses.iitm.ac.in/>

Array variable

integer array

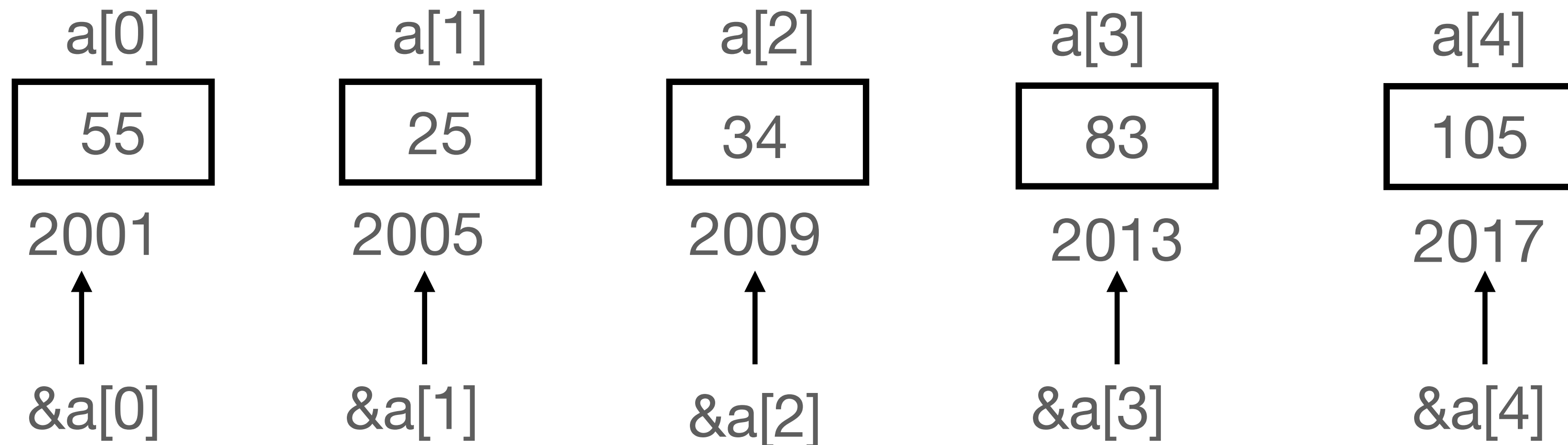
- `int a[5];`



Fetching the address of each array element

integer array

- `int a[5];`



Array variable

Better representation

- `int a[5];`

a[0]	a[1]	a[2]	a[3]	a[4]
55	25	34	83	105
2001	2005	2009	2013	2017

Array elements are always arranged in a contiguous manner i.e. the memory location or address will be having constant interval from one to the other.

Interval determined by sizeof(int)

sizeof(datatype)

- `int a[5];`

a[0]	a[1]	a[2]	a[3]	a[4]
55	25	34	83	105
2001	2005	2009	2013	2017

Interval determined by sizeof(float)

sizeof(datatype) - L12_AddressOfArrays.c

- float b[5];

b[0]	b[1]	b[2]	b[3]	b[4]
55.5	25.5	34.5	83.6	105.2
4000	4008	4016	4024	4032

CW: Find out the sizes of the data type in your computer and post your code / output in the chat box.

Base address of an array

address of the first element!

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

- base address of the array = address of the first element = `&a[0]`
- name of the array = base address! `&a[0]` can simply be `a`!

```
printf("\n", &a[0], a)
```

Hereafter, we will use the term
'base address'
quite often.

Base address of array of floats

- float b[5];

b[0]	b[1]	b[2]	b[3]	b[4]
55.5	25.5	34.5	83.6	105.2
4000	4008	4016	4024	4032

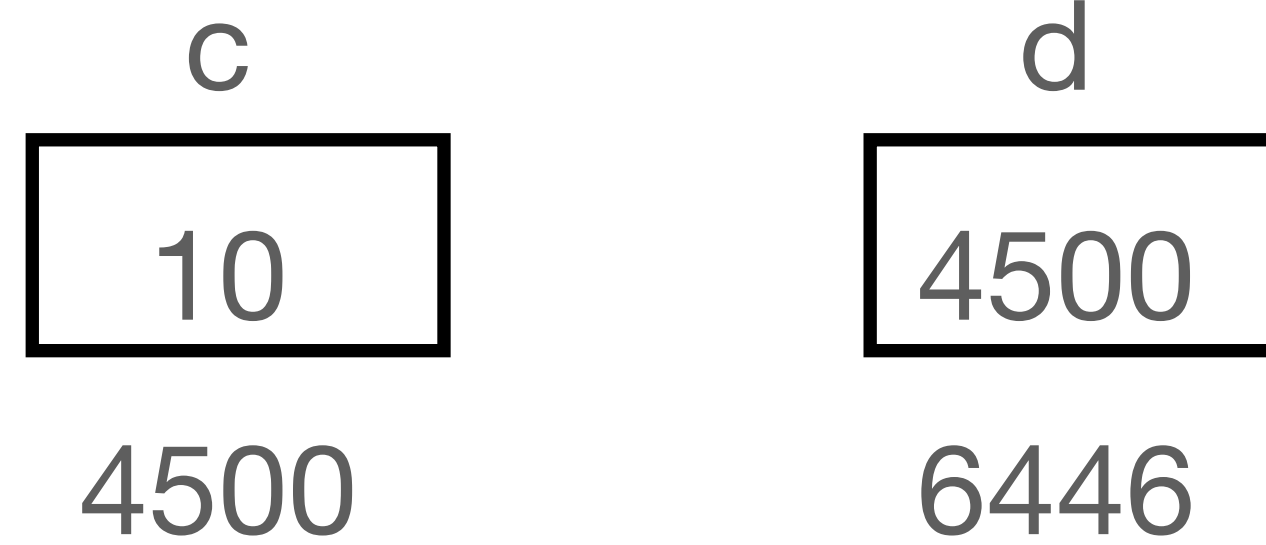
- name of the array = base address! &b[0] is simply b

```
printf("\n", b)
```


Pointer arithmetic

```
int c = 10;
```

```
int *d = &c;
```



`c + 1 = ?`

`c + 2 = ?`

What is `d + 1`?

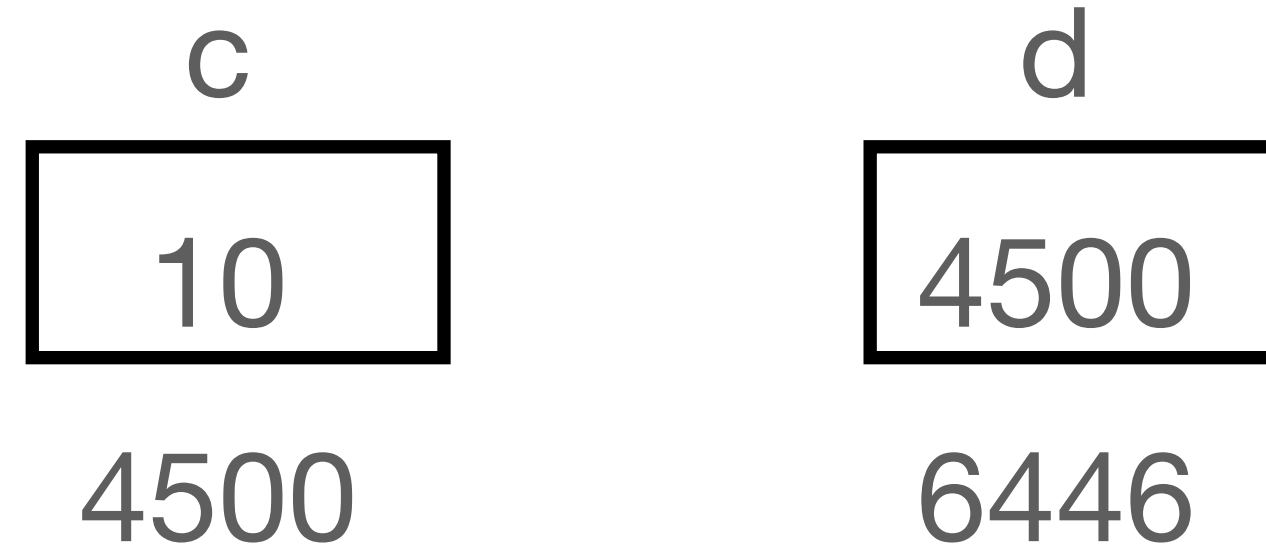
What is `d + 2`?

`d + 1 * sizeof(int)`

Pointer arithmetic

```
int c = 10;
```

```
int *d = &c;
```



$c + 2 = ?$

What is $d + 2$?

$d + 5$?

$d - 2$?

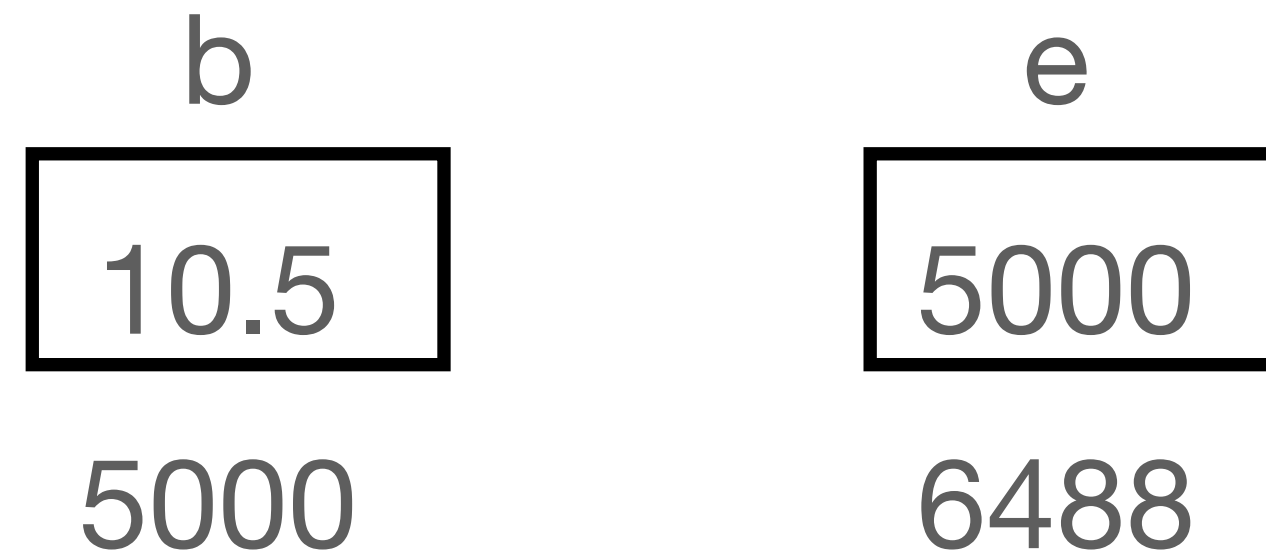
$d + 10$?

$d - 10$?

Pointer arithmetic

```
float b = 10.5;
```

```
float *e = &b;
```



$b + 1 = ?$

What is $e + 1$?

$e + 5$?

$e - 2$?

$e + 10$?

$e - 10$?

Back to integer array

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

What is `a`?

`a + 1`?

`a + 2`?

`a + 3`?

`a + 4`?

`a + 10`?

Floating point array

- `float b[5];`

b[0]	b[1]	b[2]	b[3]	b[4]
55.5	25.5	34.5	83.6	105.2
4000	4008	4016	4024	4032

What is `b`?

`b + 1`?

`b + 2`?

`b + 3`?

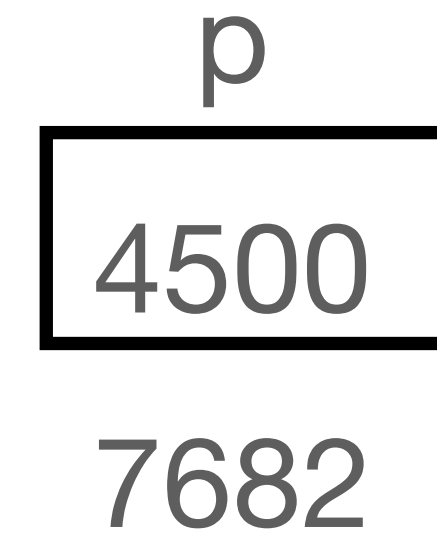
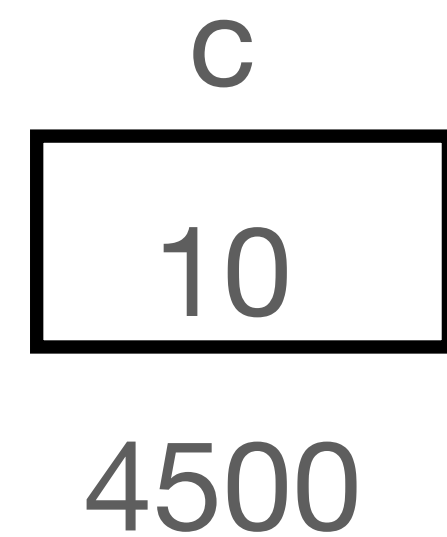
`b + 4`?

`b + 10`?

More on pointer arithmetic

```
int c = 10;
```

```
int *d = &c, *p = &c;
```



What is $d + 2$?

$p + 2$?

$d + p$?

$d * p$?

d / p ?

$d - p$?

$d / 2$?

$d * 2$?

d / p ?

Valid operations on pointer

```
int c = 10;
```

```
int *d = &c, *p = &c;
```

You can only add or subtract an integer to a pointer.

Back to integer array

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

What is `a`?

`a + 1`?

`a + 2`?

`a + 3`?

`a + 4`?

`a + 10`?

`a` implies the 'base address' of this array

When the variable is of an array type, it is essentially a pointer variable. The name of the array gives the starting (base) address.

Little more deeper!

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

WKT `a = 2001`

$*a = *(a+0) = *(2001+0) = 55 = a[0]$

Little more deeper!

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

WKT `a = 2001`

$*a = *(a+0) = *(2001+0) = 55 = a[0]$

$*(a+1) = *(2001+1) = *(2005) = 25 = a[1]$

Little more deeper!

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

WKT $a = 2001$

$*a = *(a+0) = *(2001+0) = 55 = a[0]$

$*(a+1) = *(2001+1) = *(2005) = 25 = a[1]$

$*(a+2) =$

$*(a+3) =$

Little more deeper!

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

Here is the catch!

if `*(a+2)` is valid, is `*(2+a)` valid?

Little more deeper!

- `int a[5];`

<code>a[0]</code>	<code>a[1]</code>	<code>a[2]</code>	<code>a[3]</code>	<code>a[4]</code>
55	25	34	83	105
2001	2005	2009	2013	2017

Here is the catch!

if `*(a+2)` is `a[2]`, what `*(2+a)`?

Convert scanf and printf into pointer notation

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int a[5], i;
```

```
    for (i = 0; i <= 4; i++) {
```

```
        scanf("%d", &a[i]);
```

```
        printf("%d\n", a[i]);
```

```
    }
```

```
}
```

Addition of two vectors (1D array)

Swap problem in Labwork

Function prototypes - Lab04_Swap_HW_problem.c

```
//Function prototype
void swapcv(int a1, int b1);
void swapglobal(void);
void swapcr(int *a1, int *b1);
int swapcvcr(int a1, int *b1);

int c, d; // global variables
```


Swap problem in Labwork

Functions - Lab04_Swap_HW_problem.c

```
int swapcvcr(int a1, int *b1)
{
    int c;
    c = a1;
    a1 = *b1;
    *b1 = c;

    return a1;
}
```

Swap problem in Labwork

Functions - Lab04_Swap_HW_problem.c

```
void swapglobal() {  
    int c1;  
    c1 = c;  
    c = d;  
    d = c1;  
    return;  
}
```

Swap problem in Labwork

Functions - Lab04_Swap_HW_problem.c

```
void swapcr(int *a1, int *b1) { //a1, b1 are formal arguments
    int c;
    c = *a1;
    *a1 = *b1;
    *b1 = c;
    return;
}
```

Swap problem in Labwork

Functions - Lab04_Swap_HW_problem.c

```
void swapcv(int a1, int b1) { //a1, b1 are formal arguments
    int c;
    c = a1;
    a1 = b1;
    b1 = c;
    return;
}
```

Swap problem in Labwork

Function calls - Lab04_Swap_HW_problem.c

```
swapcv(a, b);  
swapglobal();  
a = swapcvcr(a, &b);  
b = swapcvcr(b, &a);  
swapcr(&a, &b);
```

static variable

L12_StaticPointer.c

```
void demoStatic(void)
{
    static int i = 0;
    printf("The value and address of i = %d and %u\n", i, &i);
    i++;
}
```

static pointer variable

L12_StaticPointer.c

```
void demoStaticPtr(void)
{
    static int j = 0;
    static int *p = &j;
    printf("The address of p and j = %u, %u \n", p, &j);
    p++;
}
```

Passing arrays as function arguments

Back to integer array!

```
int main() {  
    int a[5] = {1, 2, 3, 4, 5};  
    for (i = 0; i < 5; i++)  
        printf("%d\n", *(a+i));  
}
```

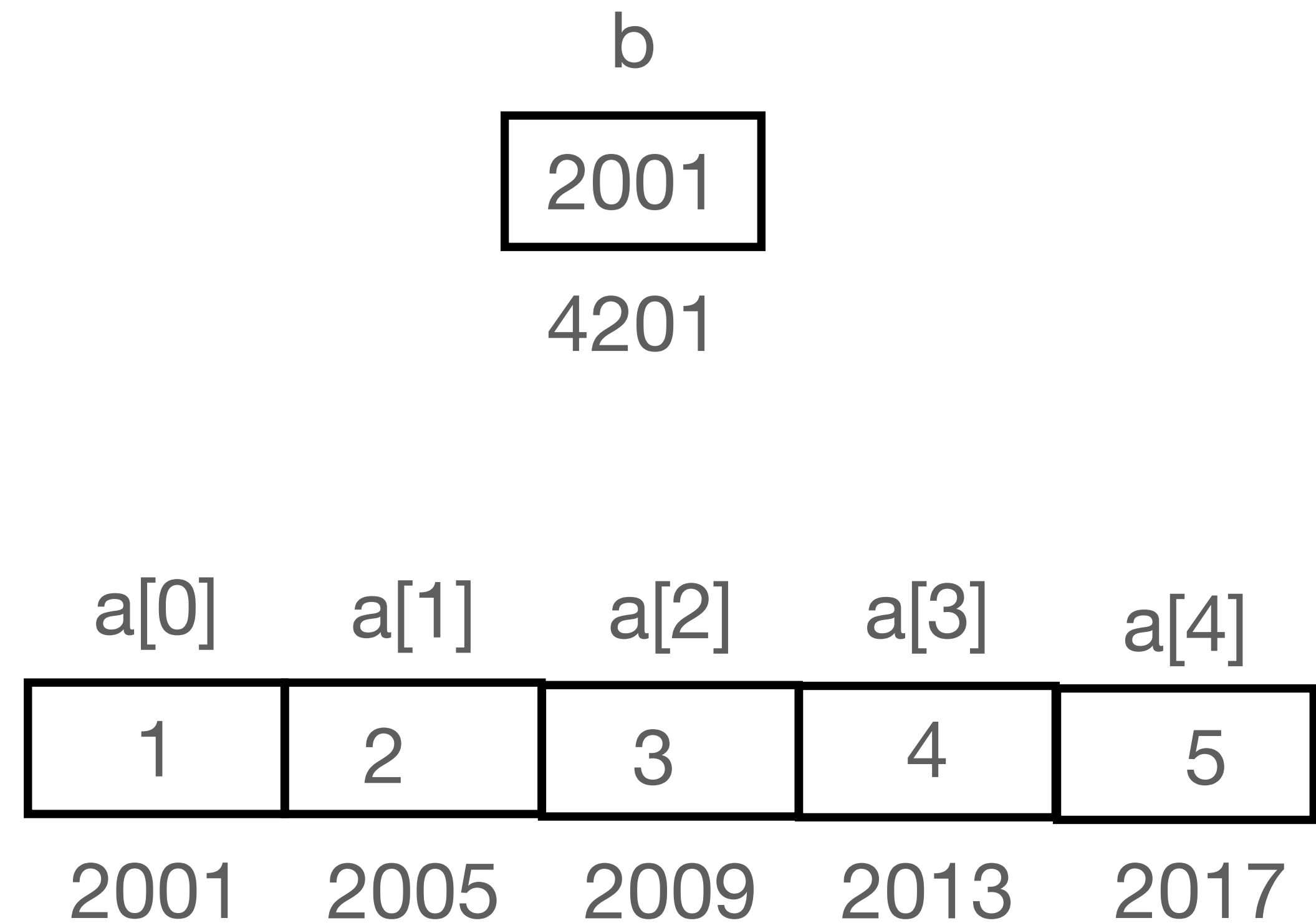
a[0]	a[1]	a[2]	a[3]	a[4]
1	2	3	4	5
2001	2005	2009	2013	2017

Passing arrays as function arguments

Essentially passing the pointer

```
PassArray(int *b)
{
    printf("%u\n", b);
}

int main() {
    int a[5] = {1, 2, 3, 4, 5};
    for (i = 0; i < 5; i++)
        printf("%d\n", *(a+i));
    PassArray(a);
}
```

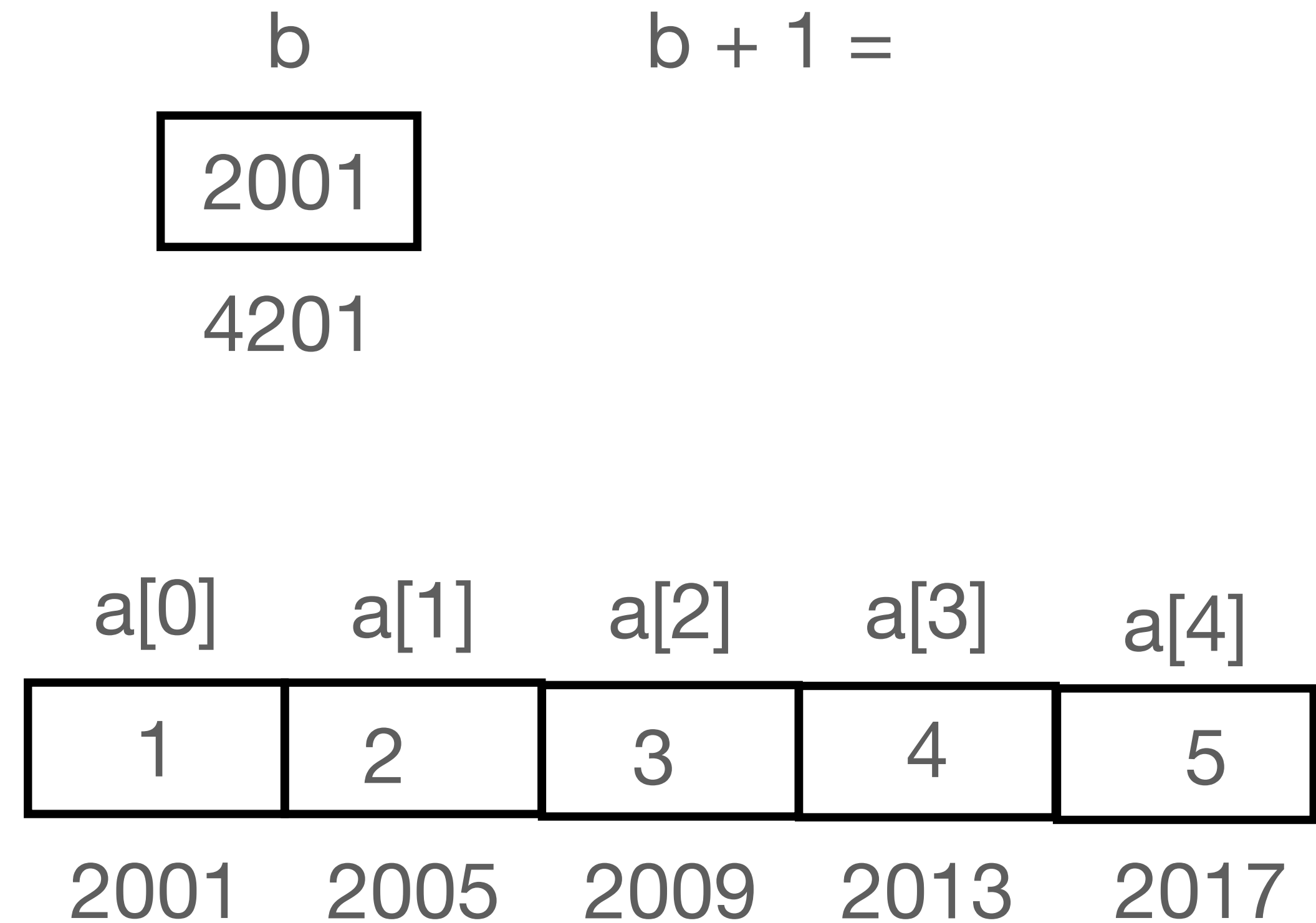


Passing arrays as function arguments

Essentially passing the pointer

```
PassArray(int *b)
{
    printf("%u\n", b);
}

int main() {
    int a[5] = {1, 2, 3, 4, 5};
    for (i = 0; i < 5; i++)
        printf("%d\n", *(a+i));
    PassArray(a);
}
```

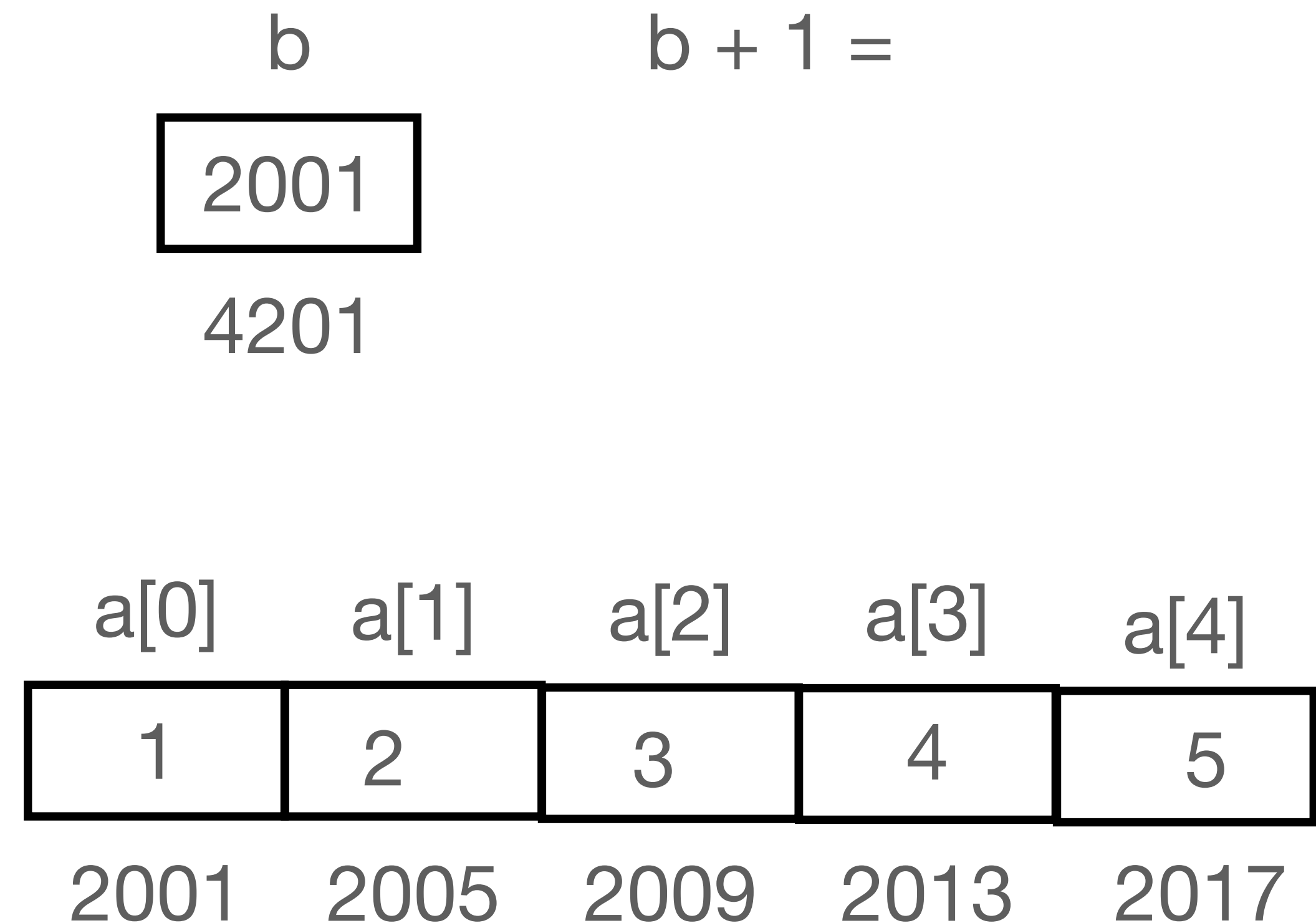


Passing arrays as function arguments

Essentially passing the pointer

```
PassArray(int *b)
{
    int i;
    printf("%u\n", b);
    for (i = 0; i < 5; i++)
        printf("%d\n", *(b+i));
}

int main() {
    int a[5] = {1, 2, 3, 4, 5};
    for (i = 0; i < 5; i++)
        printf("%d\n", *(a+i));
    PassArray(a);
}
```

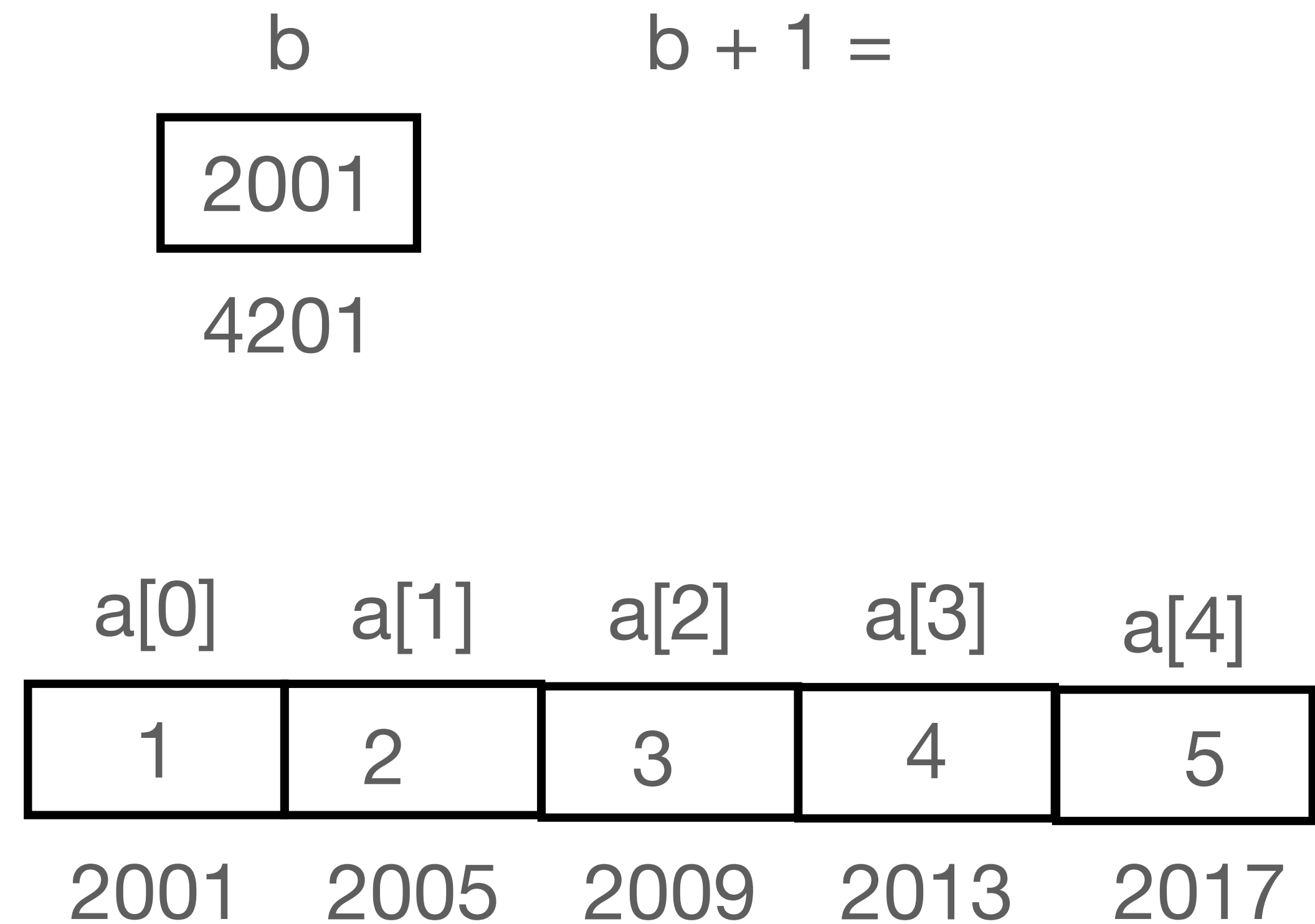


Passing arrays as function arguments

Change the value for the array in the function

```
PassArray(int *b)
{
    int i;
    printf("%u\n", b);
    for (i = 0; i < 5; i++)
        printf("%d\n", *(b+i));
    *(b+1) = 100;
}

int main() {
    int a[5] = {1, 2, 3, 4, 5};
    for (i = 0; i < 5; i++)
        printf("%d\n", *(a+i));
    PassArray(a);
}
```



Passing arrays as function arguments

Change the value for the array in the function

```
PassArray(int *b)
```

```
{  
    int i;  
    printf("%u\n", b);  
    for (i = 0; i < 5; i++)  
        printf("%d\n", *(b+i));  
    *(b+1) = 100;  
}
```

```
int main( ) {  
    int a[5] = {1, 2, 3, 4, 5, i};  
    for (i = 0; i < 5; i++)  
        printf("%d\n", *(a+i));  
    PassArray(a);  
    for (i = 0; i < 5; i++)  
        printf("%d\n", *(a+i));  
}
```

b

2001

4201

$*(b + 1) = *(2001 + 1)$
 $= *(2005)$
 $= 100$

a[0]

a[1]

a[2]

a[3]

a[4]

1

100

3

4

5

2001

2005

2009

2013

2017