4. POINTER

Nilai variable x

```
int x = 5;

Printf(" %d ", x);

Printf(" %x ", &x);
```

```
Output : 5
```

```
Output : 62fe1c
```

address variable x

```
int x = 5;
```

RAM

Address	content
. . .	. . .
62fe1c	5
. . .	. . .

Syntax for declaring pointer variables

General syntax for declaring pointer variables:

```
data_type *pointer_name
```

Here data type refers to the type of the value that the pointer will point to.

For example:

int *ptr;	← Points to integer value
char *ptr;	← Points to character value
float *ptr;	← Points to float value

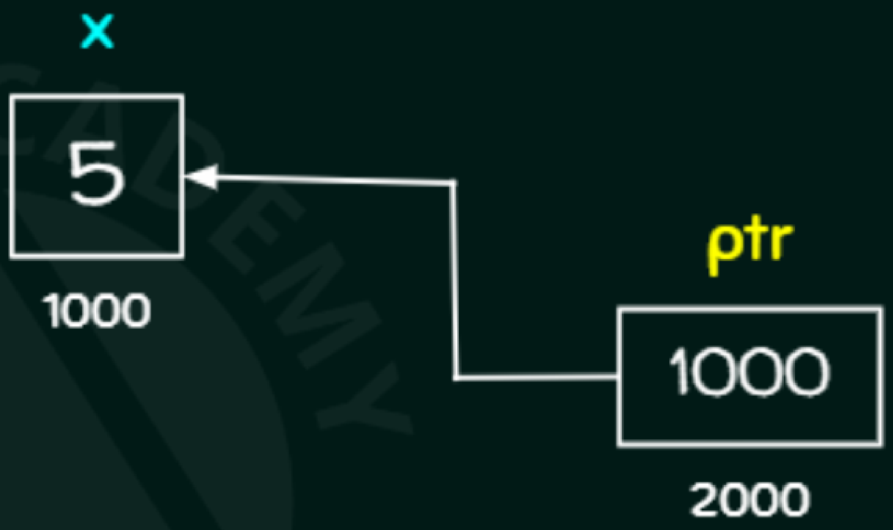
Lets try to understand how to initialize a pointer variable

Need of address of operator

- Simply declaring a pointer is not enough.
- It is important to initialize pointer before use.
- One way to initialize a pointer is to assign address of some variable.

```
int x = 5;

int *ptr;

ptr = &x;
```

address of
operator



```
int x = 5;

int *ptr;

ptr = &x;
```

}

We can also write all these lines one single line.

```c
int x = 5;

int *ptr;

ptr = &x;
```

is equivalent to

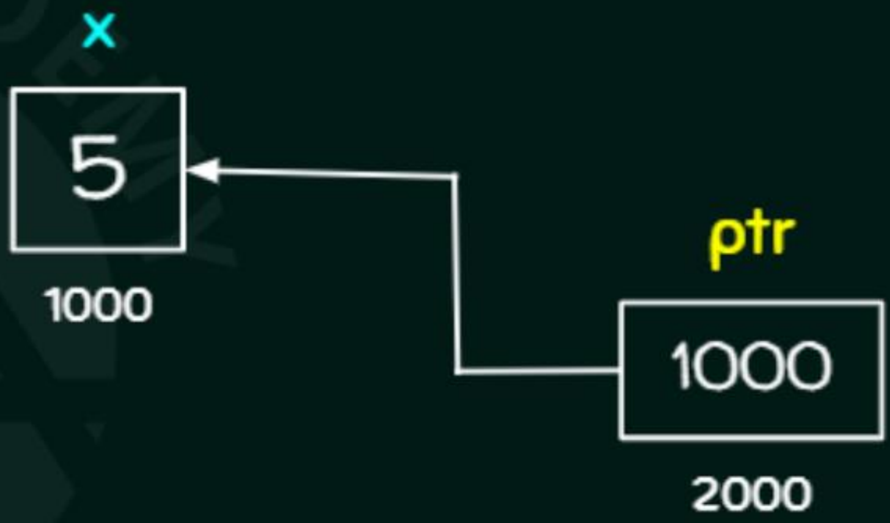
```c
int x = 5, *ptr = &x;
```

Value of operator/indirection operator/dereference operator is an operator that is used to access the value stored at the location pointed by the pointer.

```
int x = 5;

int *ptr;

ptr = &x;

printf("%d", *ptr);
```

It says go to the address of object and take what is stored in the object

VALUE OF OPERATOR

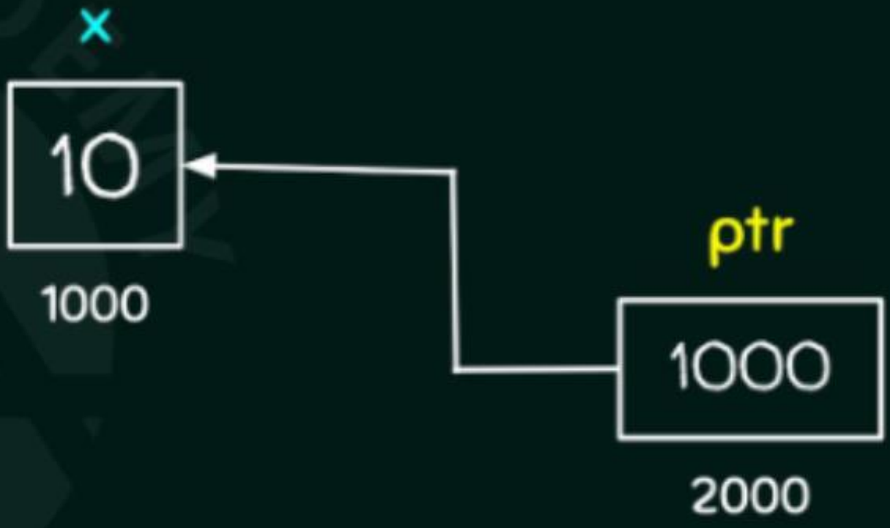


```
Output: 5
```

We can also change the value of the object pointed by the pointer.

For example:

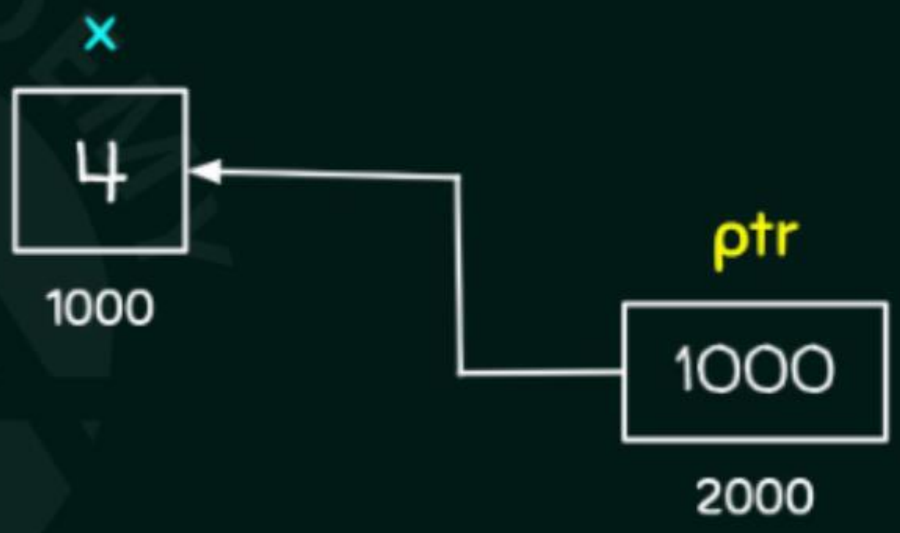
```
int x = 10;
int *ptr = &x;
```



We can also change the value of the object pointed by the pointer.

For example:

```
int x = 10;
int *ptr = &x;

*ptr = 4;

printf("%d", *ptr);
```



```
Output: 4
```

A word of caution

Never apply the indirection operator to the uninitialized pointer

For example:

```
int *ptr;
printf("%d", *ptr);
```

Output:

```
Undefined behaviour
```

One more...

Assigning value to an uninitialized pointer is dangerous.

```c
int *ptr;
*ptr = 1;
```

Output:

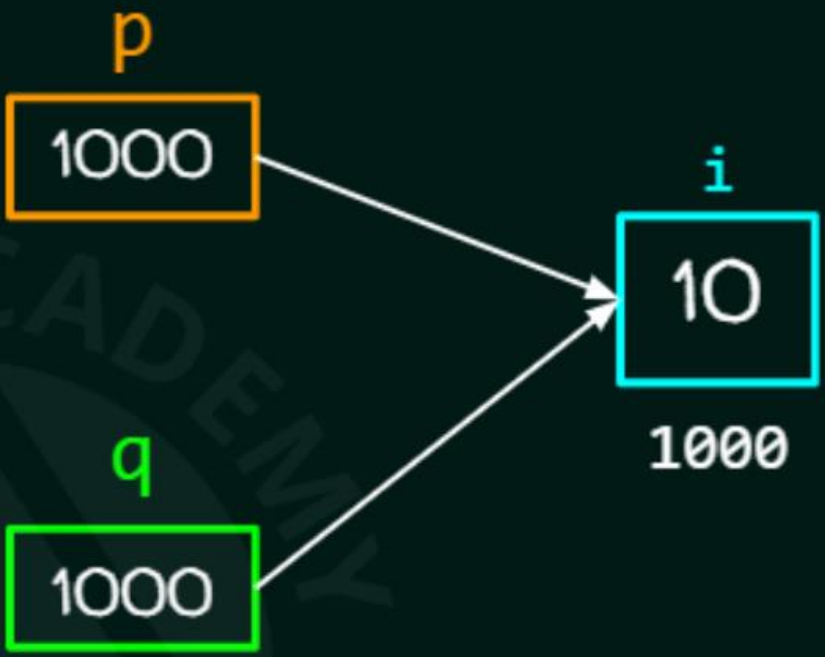
```
Segmentation Fault (SIGSEGV)
```

New Notification

Usually, segmentation fault is caused by program trying to read or write an illegal memory location.

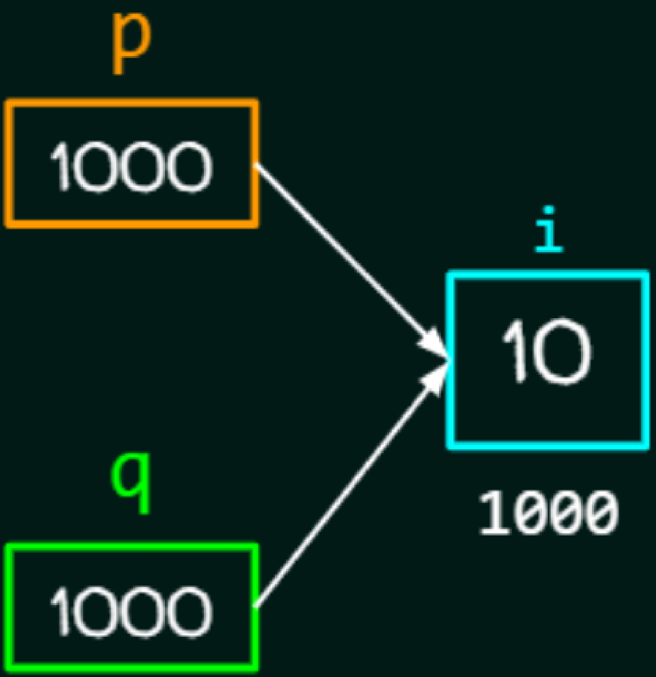
```
int i = 10;

int *p, *q;

p = &i;

q = p;

printf("%d %d", *p, *q);
```

```
Output: 10 10
```

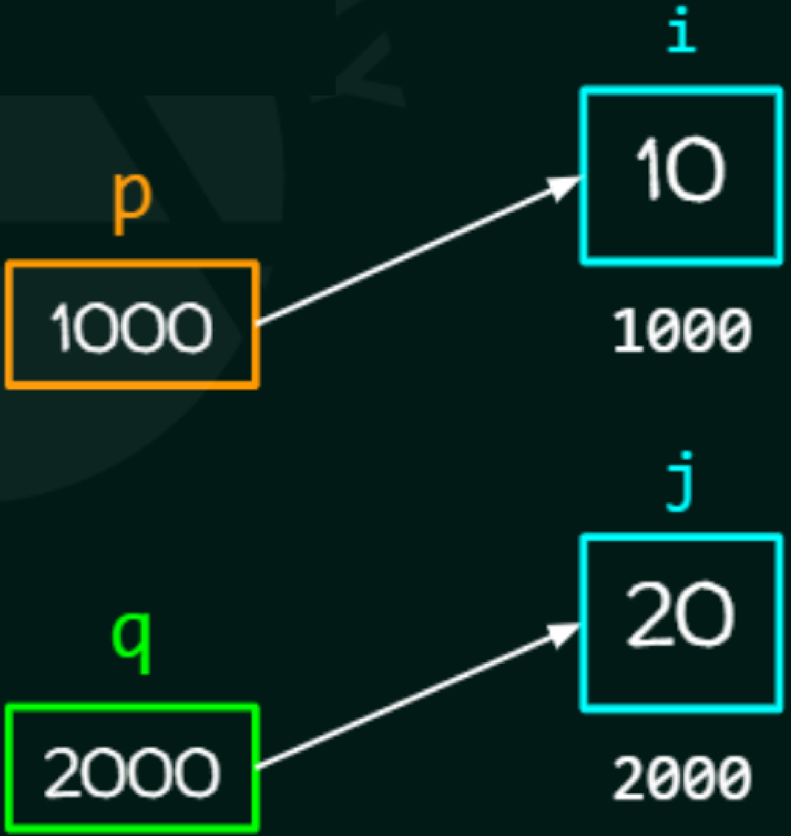


NOTE: q = p is not same as *q = *p

```
int i = 10;
int *p, *q;
p = &i;
q = p;
```

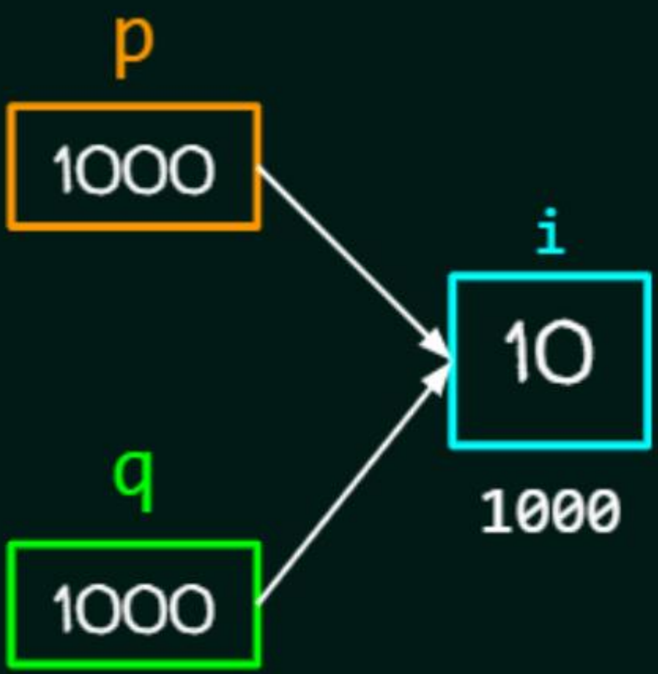


```
int i = 10, j = 20;
int *p, *q;
p = &i;
q = &j;
```

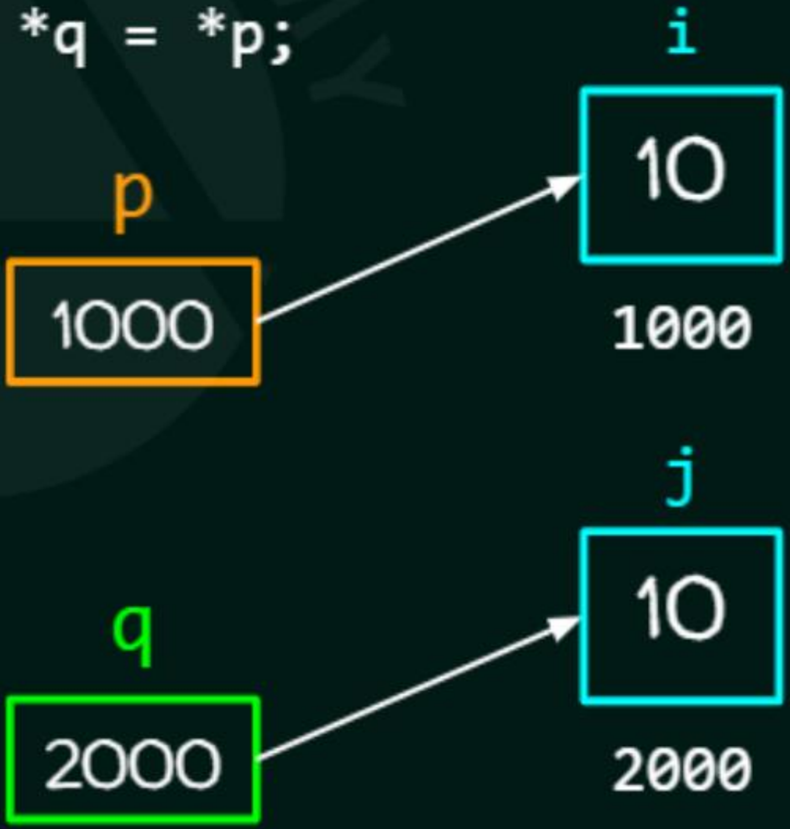


NOTE: q = p is not same as *q = *p

```
int i = 10;
int *p, *q;
p = &i;
q = p;
```



```
int i = 10, j = 20;
int *p, *q;
p = &i;
q = &j;
*q = *p;
```



Homework problem

Predict the output of the following program:

```
int i = 1;
int *p = &i;
q = p;
*q = 5;
printf(“%d”, *p);
```