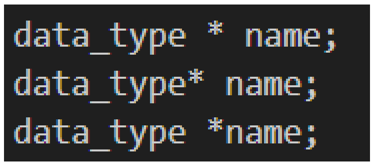
C programing language pt.7

Pointers:

Pointer is used to reference a location in the memory.

Size of pointer in the same project depending on compiler(2/4/8….).

Syntax:

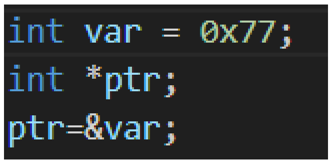


```
data_type * name;
data_type* name;
data_type *name;
```

pointer type must be the same type as that the pointer will point to.

Address operator:

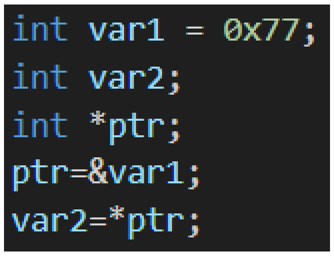
Returns the address of variable.



```
int var = 0x77;
int *ptr;
ptr=&var;
```

Dereference operator:

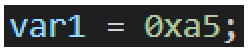
Returns the value stored in address.



```
int var1 = 0x77;
int var2;
int *ptr;
ptr=&var1;
var2=*ptr;
```

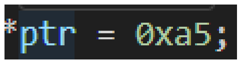
Direct and indirect access:

Direct:



```
var1 = 0xa5;
```

Indirect:



```
*ptr = 0xa5;
```

Void(generic) pointer:

Void pointer is not associated to any data type.

The important feature of void pointer is reusability.

You can’t use indirect access with void pointer before typecasting.

```
int var1 = 0x77;
int var2;
void *ptr;
ptr=&var1;
var2=*ptr; //error
var2=*(int *)ptr;
```

Wild(uninitialized) pointer:

Uninitialized pointers behavior is not defined.

Wild pointers may cause programs to crash.

We can avoid wild pointers by initializing with valid location or with NULL.

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

Dynamic memory allocation (Malloc):

Returns null if no valid memory.

```
ptr =malloc(number_of_bytes);
```

```
ptr =malloc(sizeof(data_type));
```

Free:

```
free(ptr);
```

Dangling pointer:

when the reference is deleted or deallocated without changing pointer value.

Null pointer:

Integer constant expression with value 0 or an expression cast to type void*

```
#define NULL 0
#define NULL ((void *)0)
```

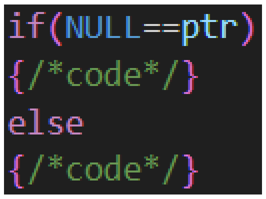
You will get a segmentation fault if you try to dereference the null pointer.

It prevents the pointer from becoming a wild pointer.

&*ptr=ptr

Two null pointers of any type will be equal.

You must validate any pointer before using it.



```
if(NULL==ptr)
{/*code*/}
else
{/*code*/}
```

Passing by value vs passing by reference:

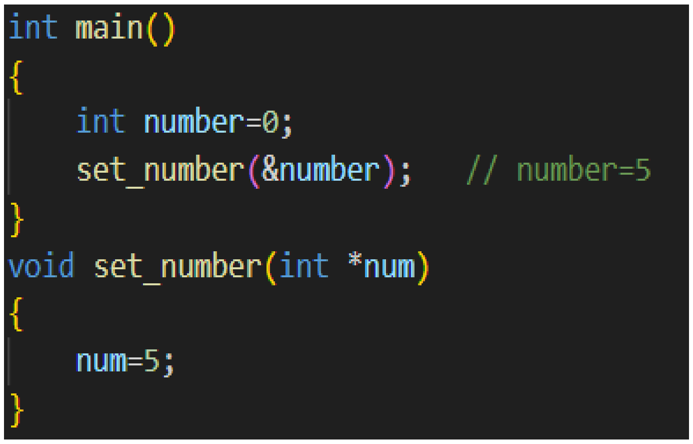
The variable doesn't change if passed by value. the effect ends when out of scope.

Calling by value affects the copy of variable.

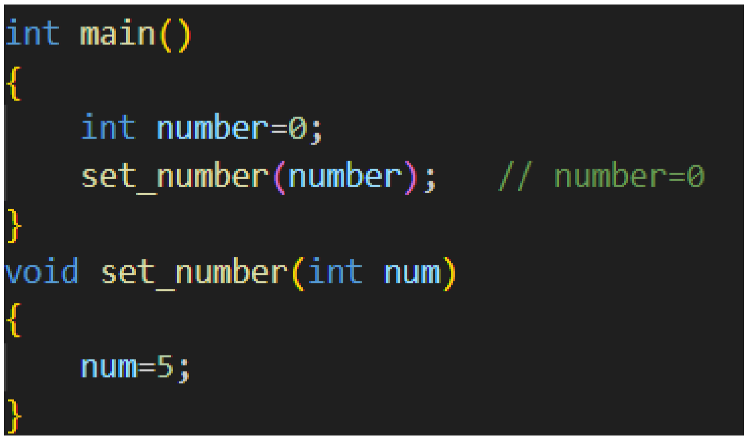
The variable doesn't change if passed by reference. the effect is permanent.

Calling by reference affects the original variable.

You pass by reference using pointers.



```
int main()
{
    int number=0;
    set_number(&number);    // number=5
}
void set_number(int *num)
{
    num=5;
}
```



```
int main()
{
    int number=0;
    set_number(number);    // number=0
}
void set_number(int num)
{
    num=5;
}
```

you can make several returns using pointers.