Pointers (1A)

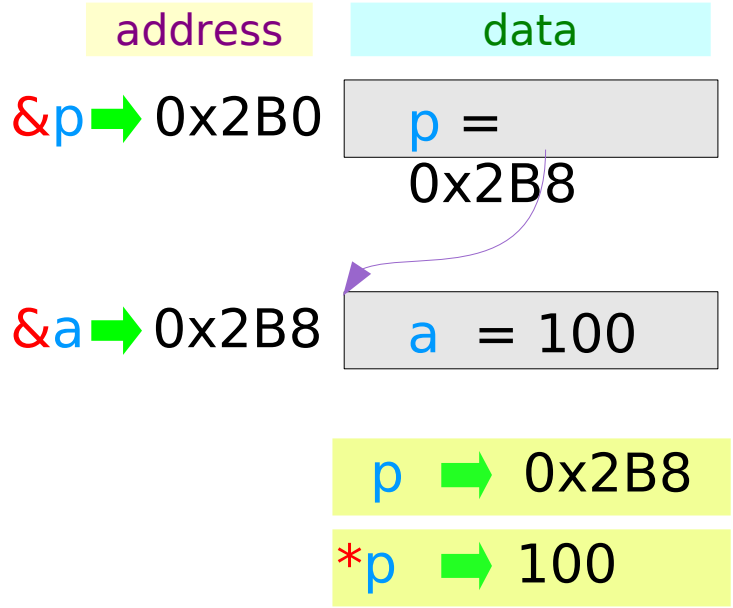
Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

* and & Operator

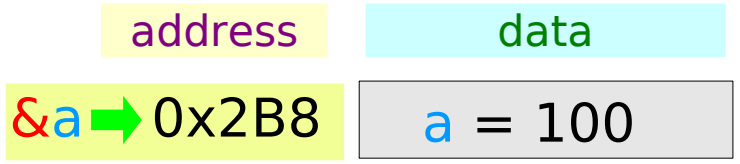
* address

returns the value
that is stored at the
address



& variable

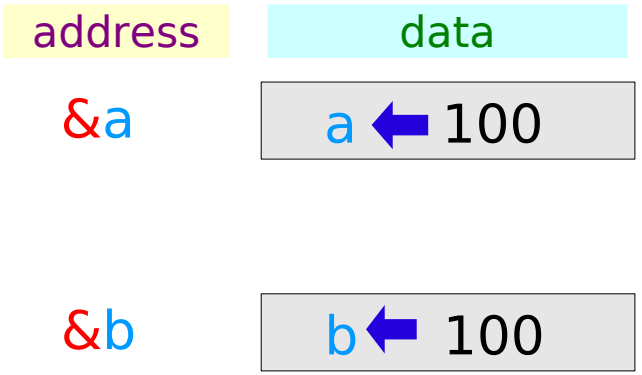
returns the address of a
location where the
variable's value is stored



Variable Initialization

```
int     a = 100 ;

int     b =  a ;
```

a can hold an *integer*
b can hold an *integer*



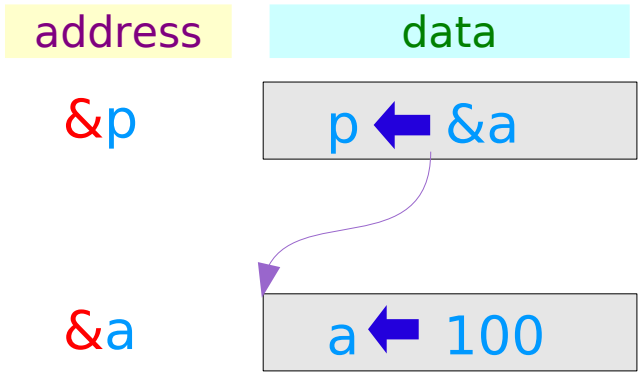
a and b have the same integer value

Pointer Variable Initialization

```
int     a = 100 ;

int *   p = & a ;
```

p can hold an *address*

p is initialized with the *address* of the integer variable a



a and *p have the same integer value, since &a and p have the same address

Reference Variable Initialization (1)

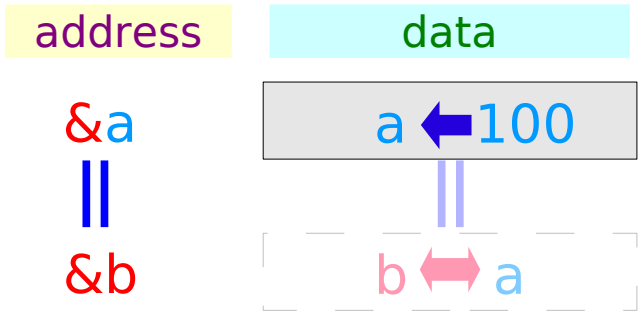
```
int     a = 100 ;
int &   b = a ;
```

b is of type "**reference to int**"

b's address is initialized with a's address

b acts like an integer variable

b holds an *integer*



&b = &a ➡ b = a

think the variable b as an ***alias*** of a

a and b have the same integer value, since &a and &b have the same address

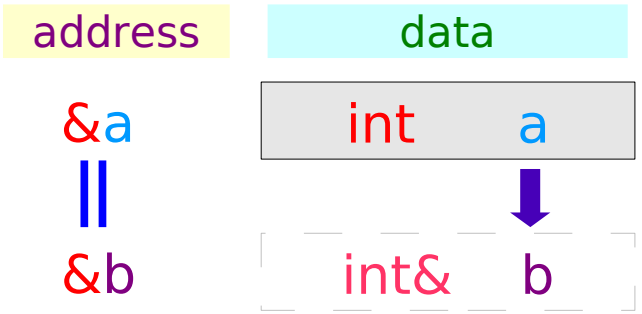
Reference Variable Initialization (2)

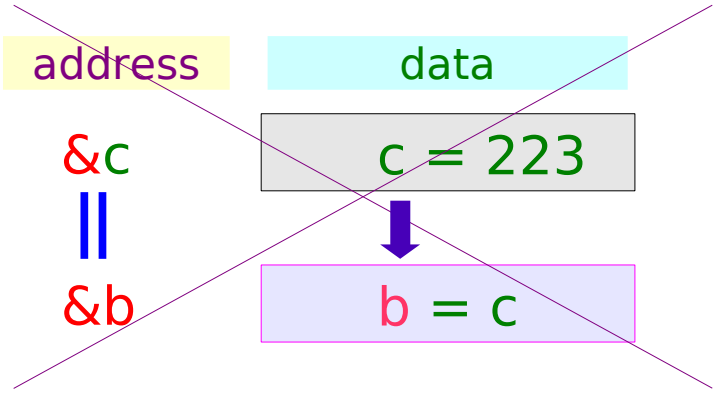
Reference is not like an ordinary variable

```
int     a = 100 ;

int &   b = a ;
```

reference:
must be initialized
cannot be changed later on

uninitialization & **reseat**
are not possible





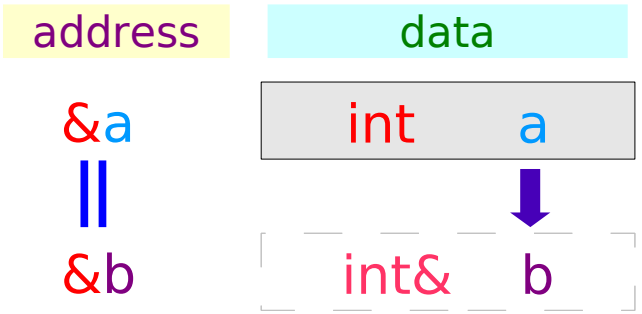
reseat is not possible

Reference Variable Initialization (3)

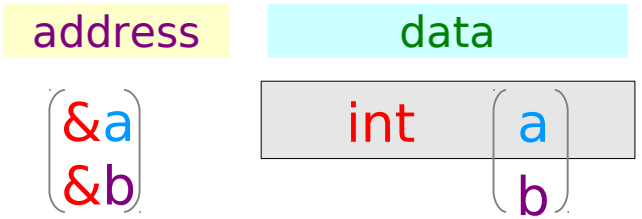
```
int     a = 100 ;

int &   b = a ;
```

Reference is not like an ordinary variable

change the state of the referent

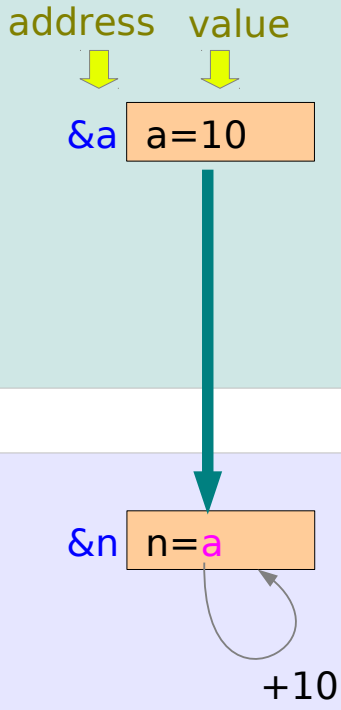


the state of the referent



Call by Value

```
void func(int n);

int main (void)
{
        int a = 10;

        printf("a = %d \n", a);
        func ( a );
        printf("a = %d \n", a);

        return 0;
}

void func (int n)
{
        printf("n = %d \n", n);
        n += 10;
        printf("n = %d \n", n);
}
```

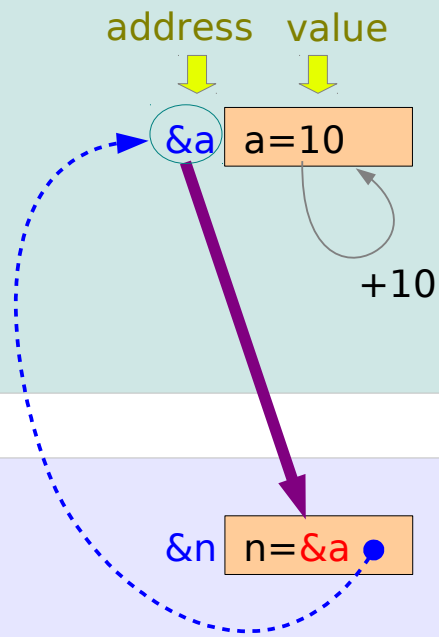


the **value** of **a** is passed through the parameter variable **n**

n is **local** to the function func and exists **while** the function is being **called**

Call by Reference – C Style

```
void func(int * n);

int main (void)
{
        int a = 10;

        printf("a = %d \n", a);
        func ( &a );
        printf("a = %d \n", a);

        return 0;
}
```



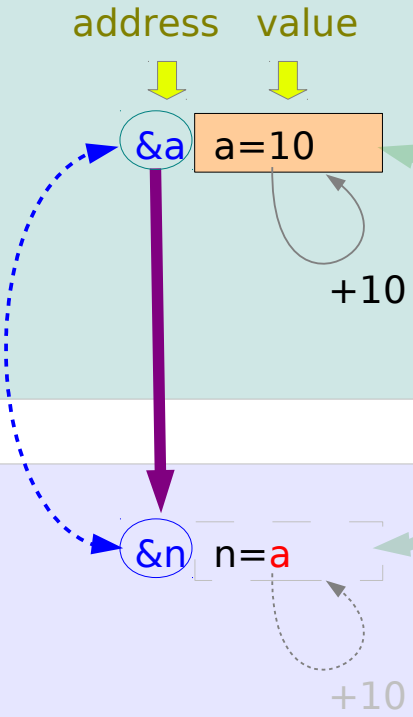
the **address** of **a** is passed through the parameter variable **n**

```
void func(int * n)
{
        printf("*n = %d \n", *n);
        *n += 10;
        printf("*n = %d \n", *n);
}
```

n is **local** to the function func and exists **while** the function is being **called**

Call by Reference – C++ Style

```
void func(int& n);

int main (void)
{
        int a = 10;

        printf("a = %d \n", a);
        func ( a );
        printf("a = %d \n", a);

        return 0;
}

void func(int& n)
{
        printf("n = %d \n", *n);
        n += 10;
        printf("n = %d \n", *n);
}
```



the **address** of **a** is passed through the parameter variable **n**

n += 10;

n is **local** to the function func and exists **while** the function is being **called**

References

[1] W Savitch, “Absolute C++”
[2] P.S. Wang, “Standard C++ with objected-oriented programming”
[3] http://www.cplusplus.com