

Pointers (1A)

Copyright (c) 2011-2013 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

This document was produced by using OpenOffice.

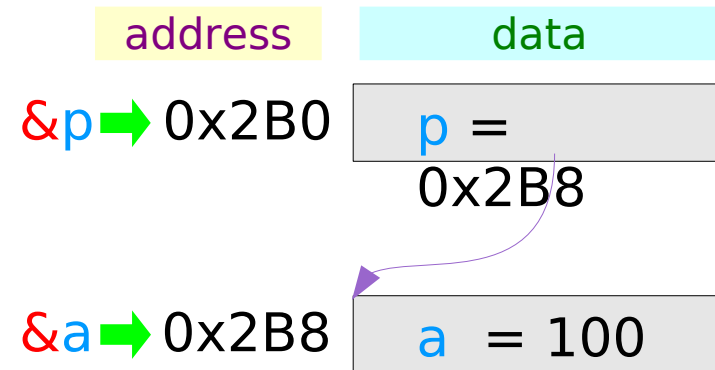
* and & Operator

* address

returns the **value**
that is stored at the
address

& variable

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



p → 0x2B8

*p → 100



Variable Initialization

int

a = 100 ;

int

b = a ;

a can hold an integer

b can hold an integer

address

data

&a

a ← 100

&b

b ← 100

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

address

data

&p

p ← &a

&a

a ← 100

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

address

data

&a

||

&b

a ← 100

b ↔ a

&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)

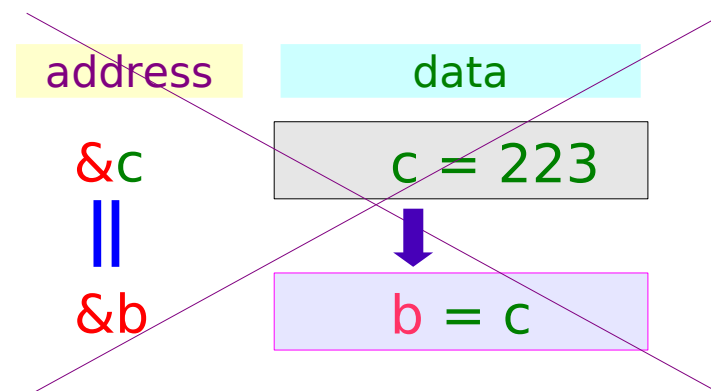
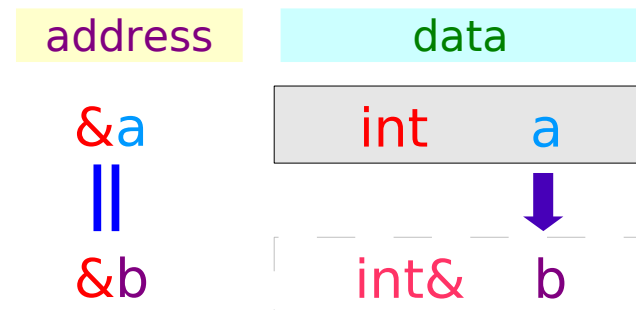
Reference is not like an ordinary variable

```
int a = 100 ;
```

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

reference:
must be initialized
cannot be changed later on

uninitialization & resear
are not possible



resear is not possible

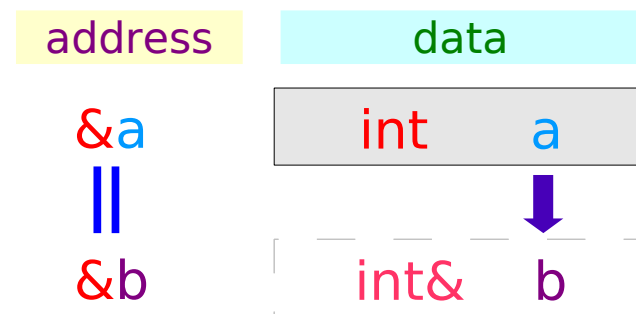
Reference Variable Initialization (3)

```
int a = 100 ;
```

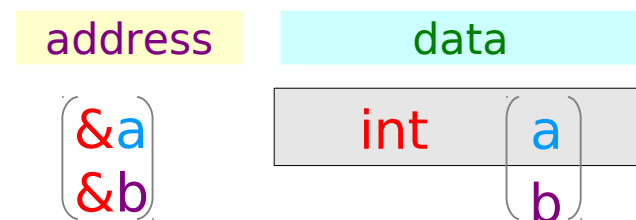
```
int & b = a ;
```

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;  
}
```

address value
↓ ↓
&a a=10

the **value** 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 n=a
 ↻
 +10

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;  
}
```

address value

&a a=10

+10

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

*n +=

10;

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

&n n=&a

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;  
}
```

address value

&a a=10

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

+10

n += 10;

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

&n n=a

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

+10

References

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