lvalues, rvalues and pointers

Designed, written and illustrated by Dmitry Vostokov

lvalues and rvalues

- lvalue – writable memory location; has an address

int a;

- rvalue – data at readable memory location or readonly value that doesn't have an address

Transient (temporary) variable (register, means that we cannot change it's value in C/C++, only to fetch)

constant (including addresses)

5 = a; // error C2106: '=' : left operand must be l-value

&a = 0x4000;

int *b = &(&a); // error C2102: '&' requires l-value

- All lvalues are rvalues also because writable memory location is readable also

Transient (temporary) variables

- They are rvalues, don't have permanent address
- Where expression value is a transient value

```
int a, n;
int *pa;
a = n + 2; // n+2 doesn't have an address
pa = &(n + 2); // error
a = n; // n is rvalue but not transient
a = n = a; // n is lvalue and rvalue
```

- Mnemonics: rvalue – value on the right or readonly value

Pointers

- A variable that contains memory address. It has an address, writable and therefore lvalue.
- Pointers can point to any type

Type *pname = $\{_{opt}$ initialization_expression $\}_{opt}$;

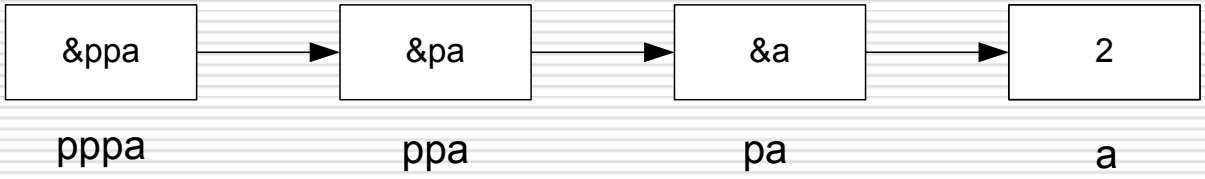
```
int a;
int *pa = &a;
int *pa = { &a };

HWND *phWnd = &hWnd;
```

- The size of the pointer is 32 bits (on x86 Windows) and 64-bit on x64 Windows

Pointers to pointers

```
int a = 2;
int *pa = &a;
int **ppa = &pa;
int ***pppa = &ppa;
```



Pointer assignment

- You can only assign pointer to pointer if they point to the same type:

```
int a; long b;
int *pa = &a;
int *pa2 = pa;

pa = &b; // error &b has ‘long *’ type
```

Dereferencing a pointer

- To access a value pointed to we use indirection (dereferencing) operator '*'

```
int a, *pa = &a;
int val = *pa;  // val == a


int **ppa = &pa;
val = **ppa;
```

Tricky example

```
int a, *pa = &a;
pa = &(*pa);
```

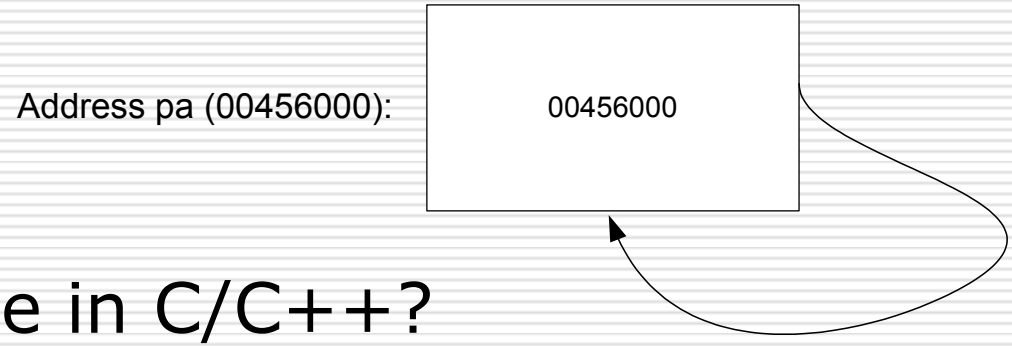
*pa is writable, we can write *pa = 1;

So it is lvalue and we can take address of it

More tricky stuff

Can the pointer contain it own address?

lea eax, pa
mov [pa], eax



Can we do the same in C/C++?

pa = &pa; // error, &pa has ‘int **’ type

Answer to previous question

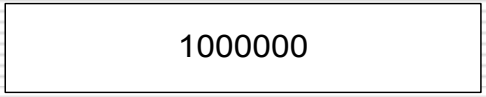
Yes, we can:

```
pa = (int *)&pa;
```

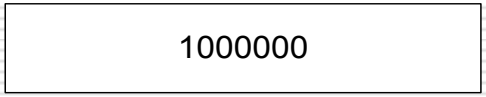
This is so called type conversion, more on this next time.

Type conversions (traditional casts)

int a;



int *pointer;

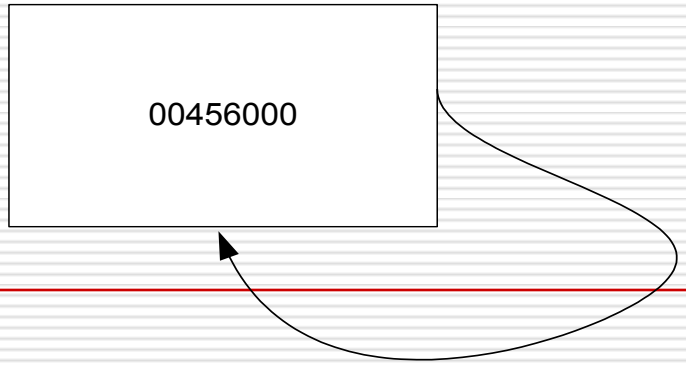



```
mov eax, dword ptr [a]
mov dword ptr [pointer], eax
```

pointer = (int *)a;

a = (int)pointer;

int *ptr;
Address ptr (00456000):



ptr = (int *)&ptr;

Pointers to const

```
const char ccA = 'A';
const char ccB = 'B';

// read declaration/definition from right to left
const char *pcc = &ccA;
*pcc = 'B';  // error   *pcc - rvalue
pcc = &ccB;
```

Const pointers

```
char ccA = 'A';
char ccB = 'B';

// read declaration/definition from right to left
char * const pcc = &ccA;
*pcc = 'B';
pcc = &ccB;  // error - rvalue
```

Const pointers to const

```
const char ccA = 'A';
const char ccB = 'B';

// read declaration/definition from right to left
const char * const pcc = &ccA;
*pcc = 'B';    // error    *pcc - rvalue
pcc = &ccB;  // error    pcc - rvalue
```

Const casts

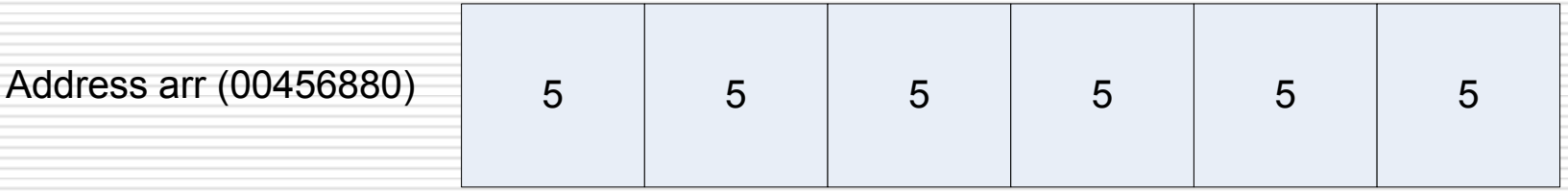
```
const char ccA = ‘A’;
const char ccB = ‘B’;

// read declaration/definition from right to left
const char * const pcc = &ccA;
*(char * const)pcc = ‘B’;
(const char *)pcc = &ccB;          // error
*&(const char *)pcc = &ccB;
```

Arrays (one-dimensional)

- Type name[size] = { init_list } ;

int values[3] = { 0, 1, 2 };



int arr[6];

arr is rvalue

int *pi = &arr[0];

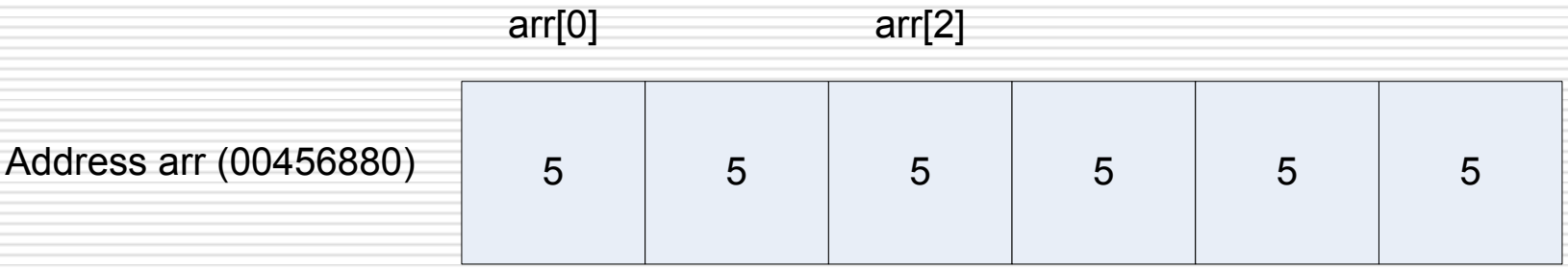
00456880

pi is lvalue

Array and pointer relationship

- Basic Rule

a[i] is equivalent to *(a + i)

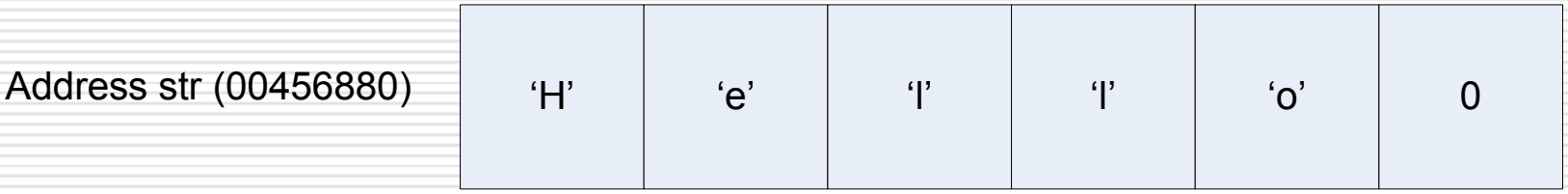


&arr[2] == arr + 2 == (int *)00456888

arr + 2 = address of arr + 2 * size of element in bytes

Strings

- char str[size];
- containes at most size-1 characters
- Empty string has str[0] == '\0'



char str[6];

str is rvalue

char *pStr = &str[0];

pStr is lvalue