Windows Programming

Lecture 02

Pointers

Today’s Lecture Agenda

- Discussion about Pointers width
- Pointer arithmetic
- Pointer-to-Pointer (A Short Introduction)

Marking rooms using binary numbers 0 and 1

Marking rooms using binary numbers 0 and 1

(2 marking places)

Marking rooms using binary numbers 0 and 1

(3 marking places)

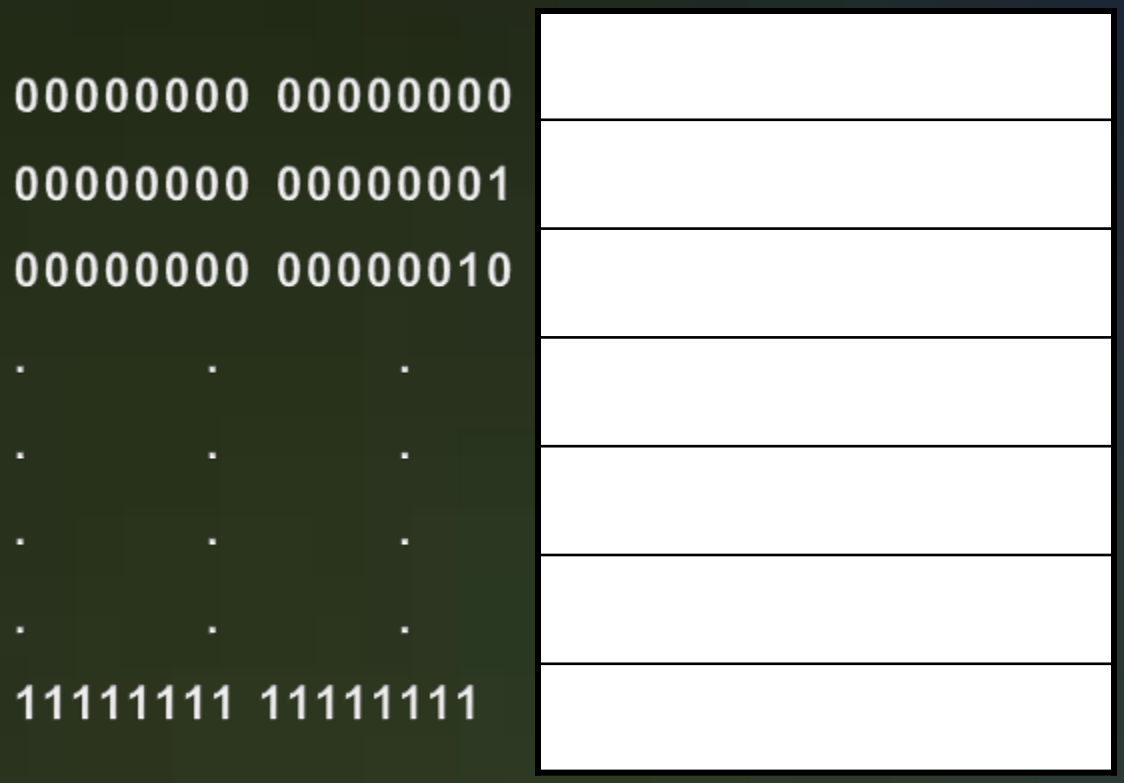
Marking Boards	No. of Rooms
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
n	2^n

RAM

(Random Access Memory)

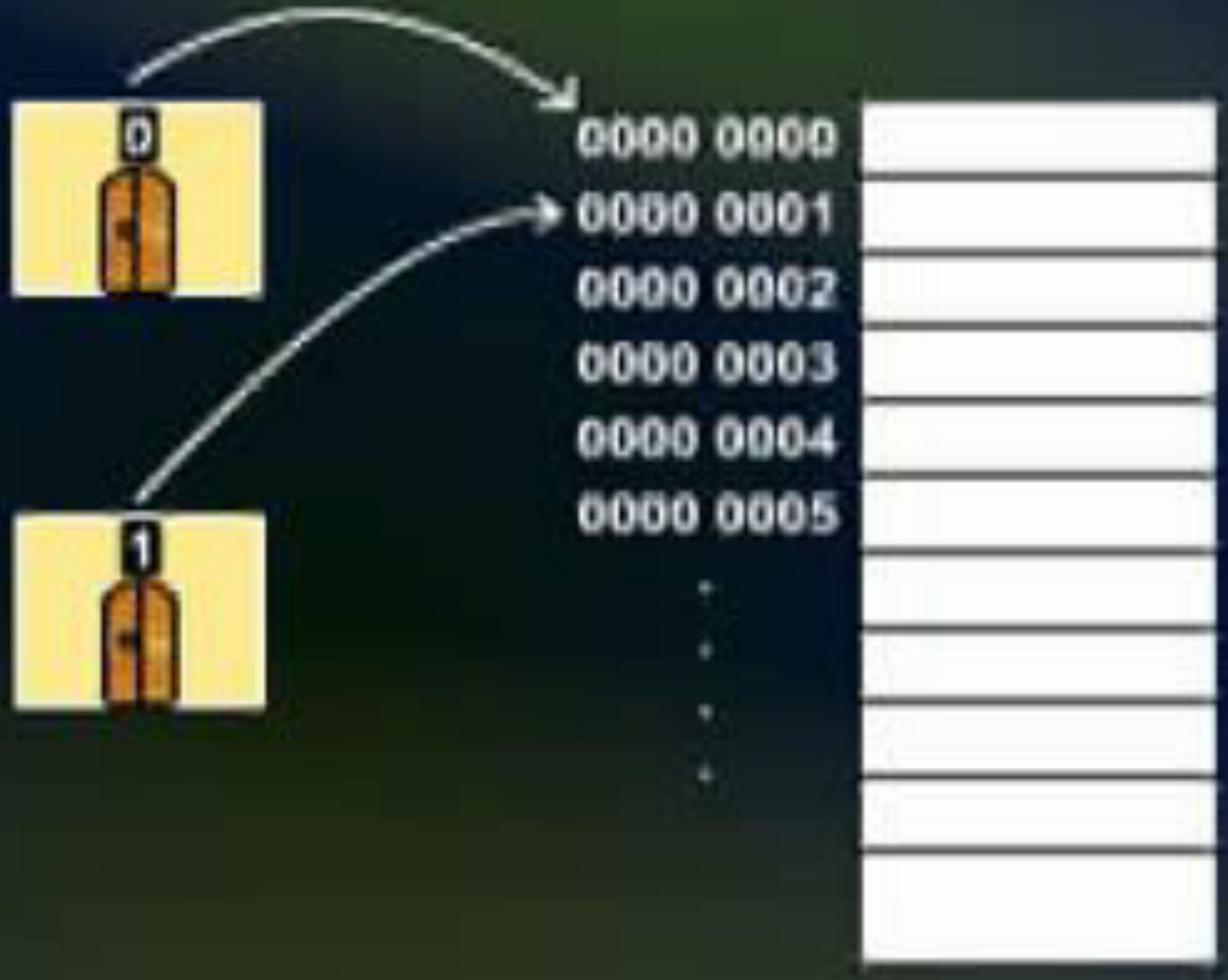
RAM (random access memory) is the place in a computer where the operating system, application programs, and data in current use are kept so that they can be quickly reached by the computer's processor.

- Every byte in Ram has an address



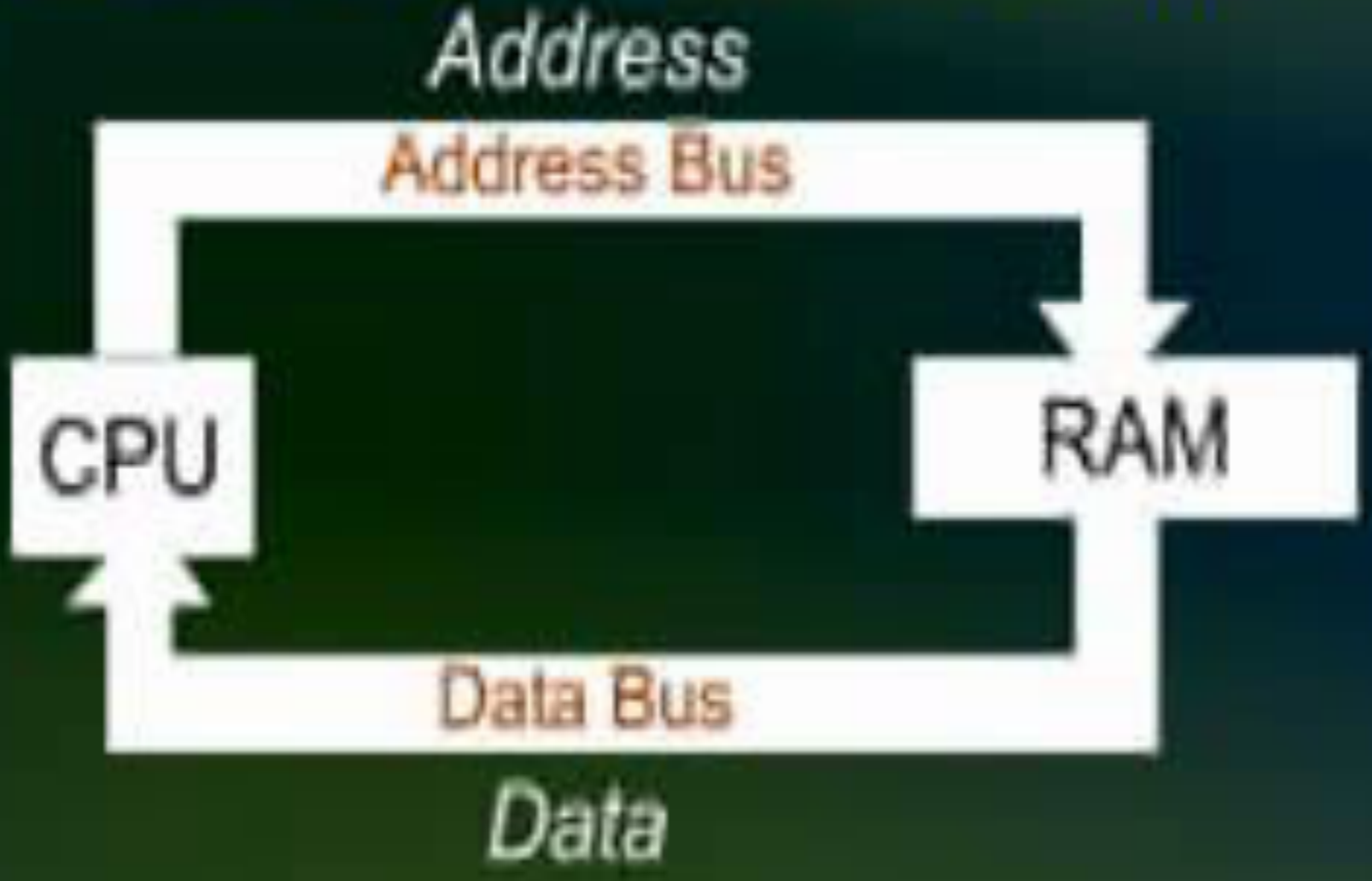
Max address = 2^{16} = 65536

= 64K


0
1
0000 0000
0000 0001
0000 0002
0000 0003
0000 0004
0000 0005

Width of Address Bus	No. of Bytes
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
n	2^n

Reading Data from RAM



00000000	00000000	= 0
00000000	00000001	= 1
00000000	00000010	= 2
00000000	00000011	= 3
.	.	.
.	.	.
.	.	.
11111111	11111110	= 65534
11111111	11111111	= 65535

Maximum addresses = 2^{16} = 65536

= **64K**

Address Bus

- Older computers had
  16-bit address bus
- Today most computers have
  32-bit address bus

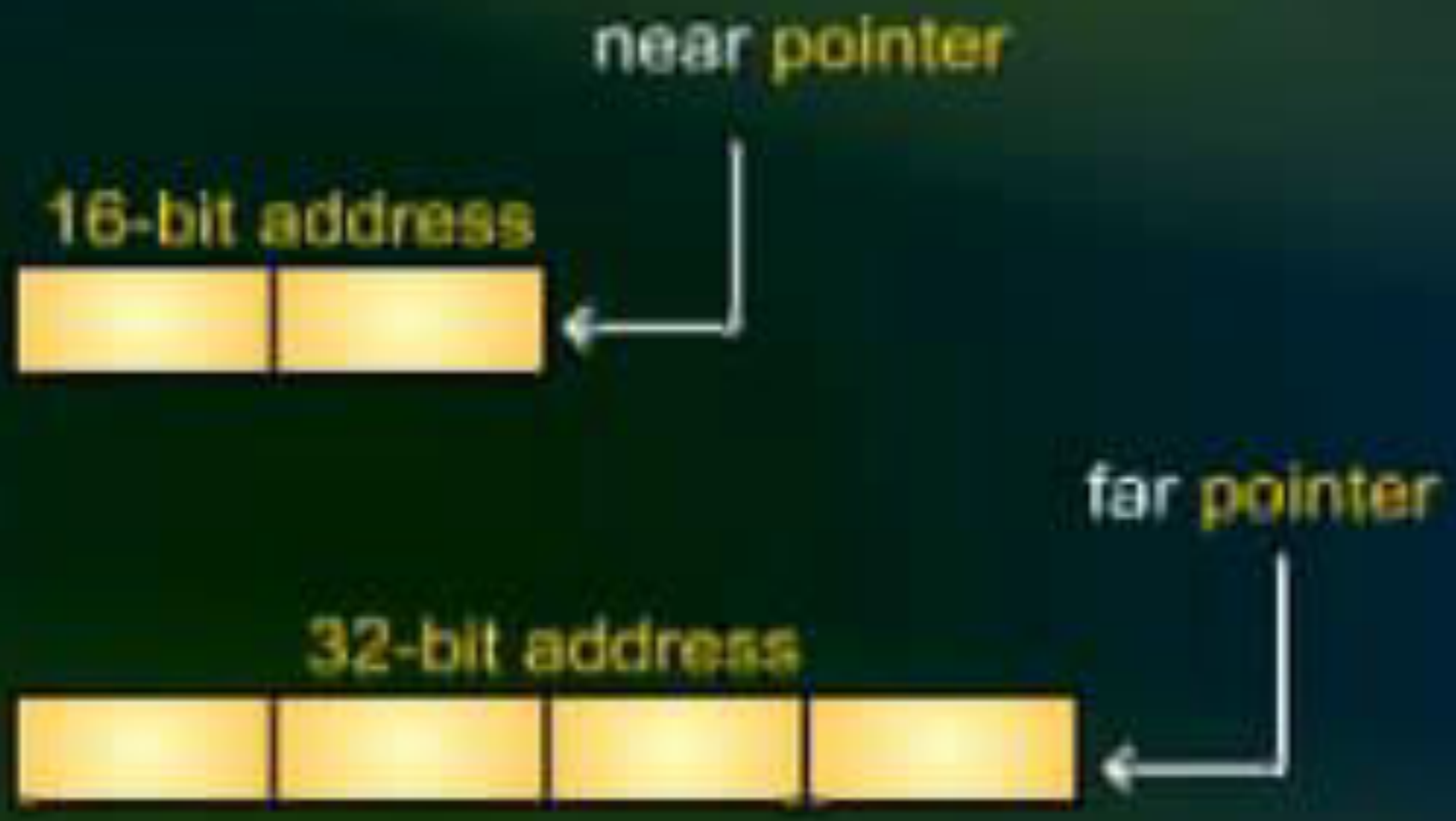
Pointer

- Is a variable which can store the address of any byte of a RAM
- Guess, if the size of address bus is 16 bits then what will be address size of each byte ??
- It's also 16-bits wide
- The size of the pointer will be of 2 bytes to store the address of bytes

Pointer

- **In Today's computers, size of address bus is 32 bits so the address size of each byte is also 32 bits .**

- **In the same way the size of the pointer will be of 4 bytes to store the address of bytes.**


near pointer
16-bit address
far pointer
32-bit address

Memory model

- All pointers are 32-bit pointers in Win32 programming.
- There is no concept of ***near*** or ***far*** pointers.
- There is only one memory model *i.e.* ***flat memory model*** in 32-bit environment

```
int i, *ptr;
```

i is an integer variable

```
int i, *ptr;
```

*ptr is a pointer to integer variable

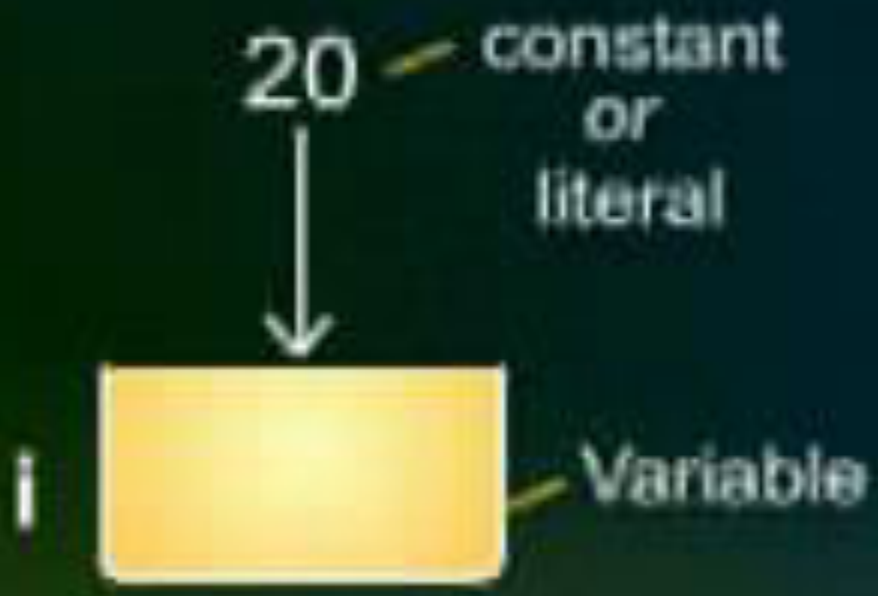
```
int i, *ptr;
```

which means that ptr is a variable which will store the address of integer variable

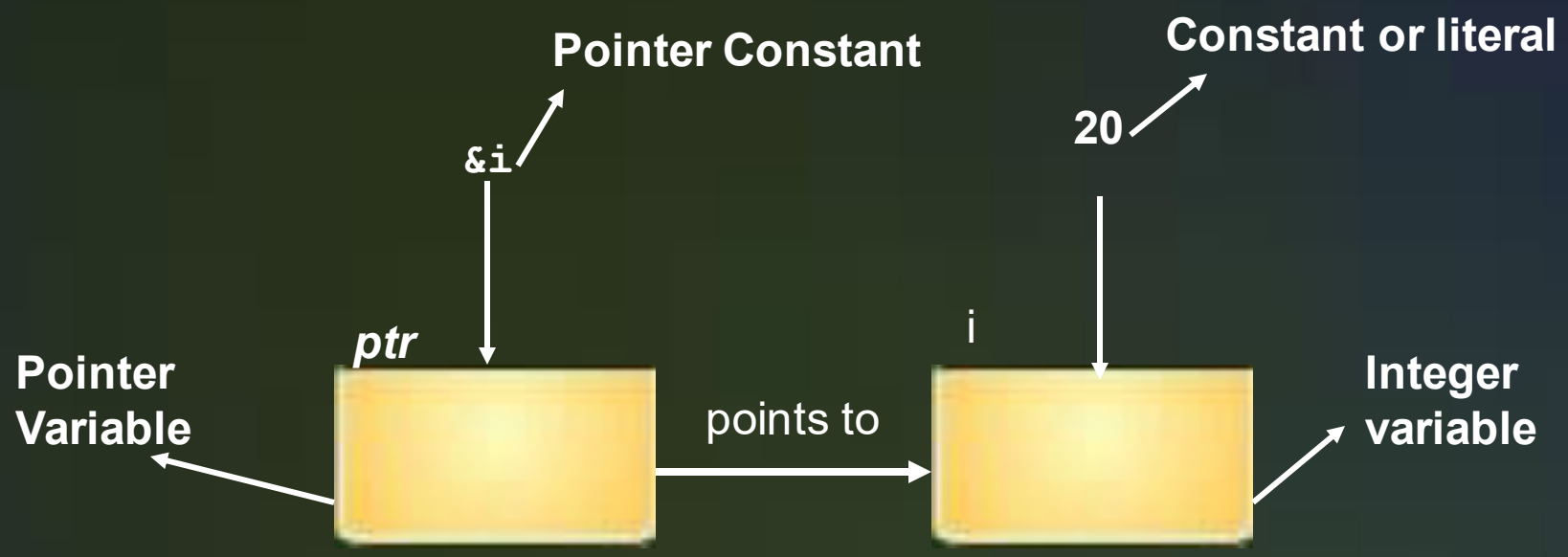
```
double *ptr;
```

which means that ptr is a variable which will store the address of double variables

```
int i = 20;
```



```
int i, *ptr;
i = 20;
ptr = &I;
```



In Windows

Double takes 8 bytes

Integer takes 4 bytes

Float takes 4 bytes

But the address size of each of them is fixed and of same size i.e. 4 bytes

Why???

Address of a variable is address of its lowest byte


207
208
209
20A
20B
20C
20D
20E
207
pointer
to double

```
int i, *ptr;

ptr = &i;
```

& is the

Reference / Address operator

- &i means that address of i
- ptr = &i; // In this statement lowest byte address of integer variable i will be assigned to ptr variable.

```
int i, *ptr;

ptr = &i;

*ptr = 4;
i = 4;
```

- Both statements *ptr = 4 and i=4 are same.
- *ptr means that what ever address is placed in ptr go to that location and store 4 at that location.

- ptr and i become alias of each other.
- Means two names of same thing
- We can use them interchangeable

```
int i, *ptr;

ptr = &i;
```

```
i = 4;
```

1 write operation

```
*ptr = 4;
```

1 read operation
1 write operation

Operators used with pointers

`( * )` Indirection / De_referencing

`( & )` Address / Reference

Indirection

- A method in which first we read the address and then go to that address and write/store on that address.
- We access the variables indirectly.

Difference between * and & operator

& operator takes the address of any variable whereas * operator go to some specific address and access that particular variable

Difference between

Int *ptr;

And

*ptr = 5;

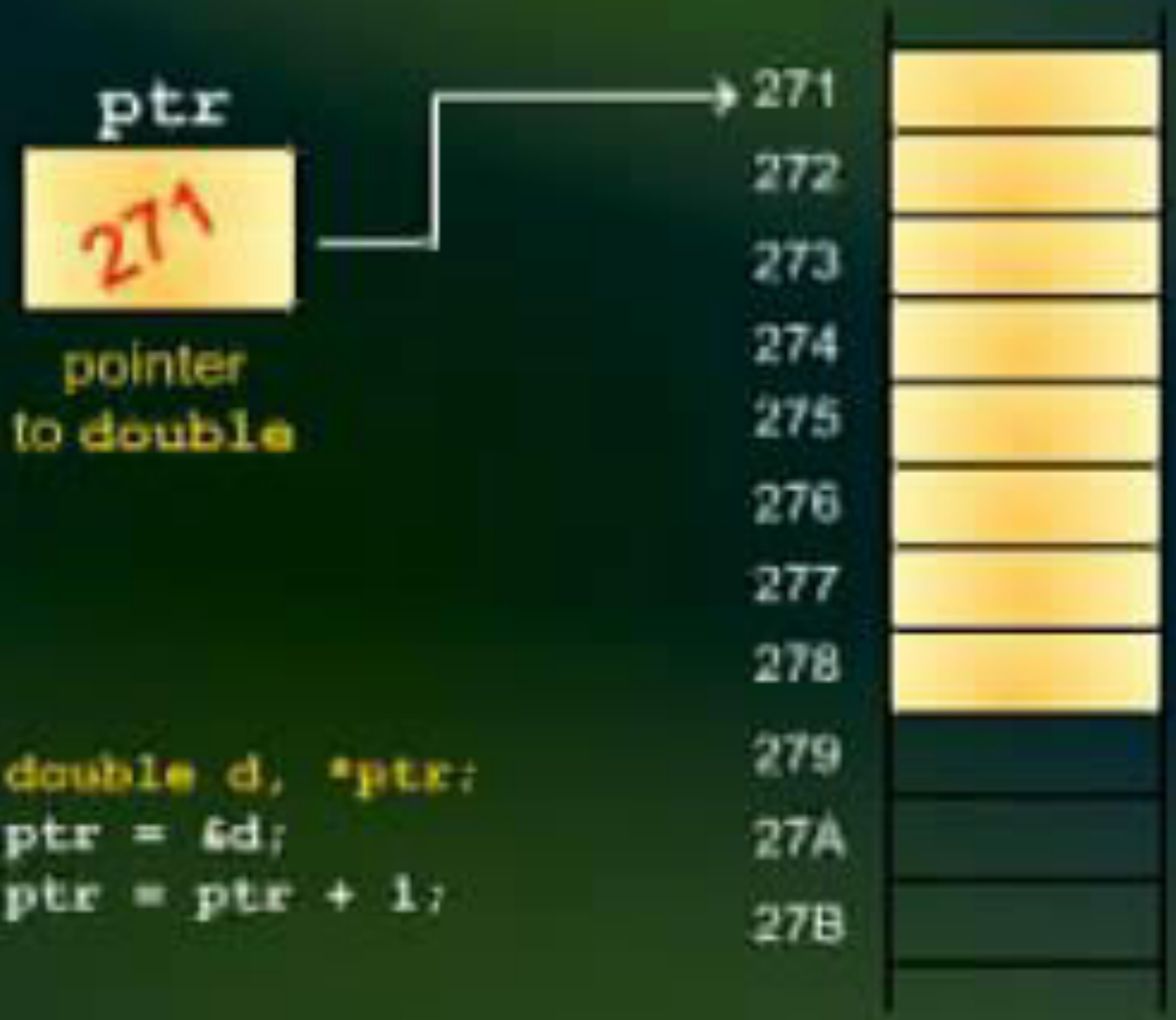
- int *ptr; // Declaration of pointer
- *ptr = 5; // Assigning a value to pointer

- Symbol is same but meaning is different
- We differentiate between them w.r.t to their context mean where they are used
- Compiler know this difference.
- Reason: Easy to remember

Pointer Arithmetic

- Two *pointers* can be added / subtracted
- An integer can be added to or subtracted from a pointer.
- Only addition(+) and subtraction(-) are possible in pointer arithmetic.

```
double d, *ptr;

ptr = &d;

ptr = ptr + 1;
```




```
double d, *ptr;
ptr = &d;
ptr = ptr + 1;
```

ptr = ptr + 1

pointer
to double


```
double d, *ptr;
ptr = &d;
ptr = ptr + 1;
```

- Address in pointer is incremented or decremented by the size of the object it points to (char = 1 byte,...)

What is the Significance of pointer data types?

```
double d, *ptr;

ptr = &d;
*ptr = 2.7;
```

*ptr modifies 8 bytes

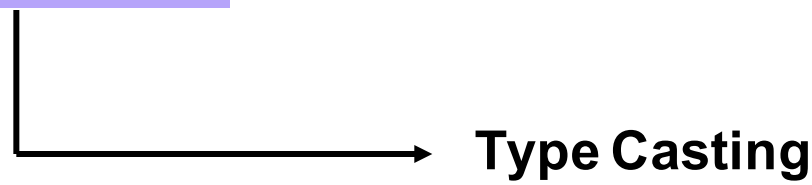
Significance of pointer data types

- How many bytes in RAM are affected by indirection ?
- By how many bytes a pointer moves backward or forward during pointer arithmetic ?

Type Casting

- A pointer of one type can be type casted into a pointer of another type;

```
float *ptr;
char *ptrc;
ptrc = (char *) ptr;
```



```
double d, *ptr;
char  *ptr2;

ptr = &d;

ptr = ptr + 1;
```

ptr skips 8 bytes

ptr skips ***1*** *byte*

```
ptr2 = (char *)ptr + 1;
```

typecast operator

Effect of pointer-typecasting

```
int i, *ptr;

ptr = &i;
```

write into the single lower byte

```
*((char *)ptr) = 'A';
```

OR

```
*((char *)ptr) = 0x41;
```

write into the single upper byte

```
*((char *)ptr + 1) = 65;
```

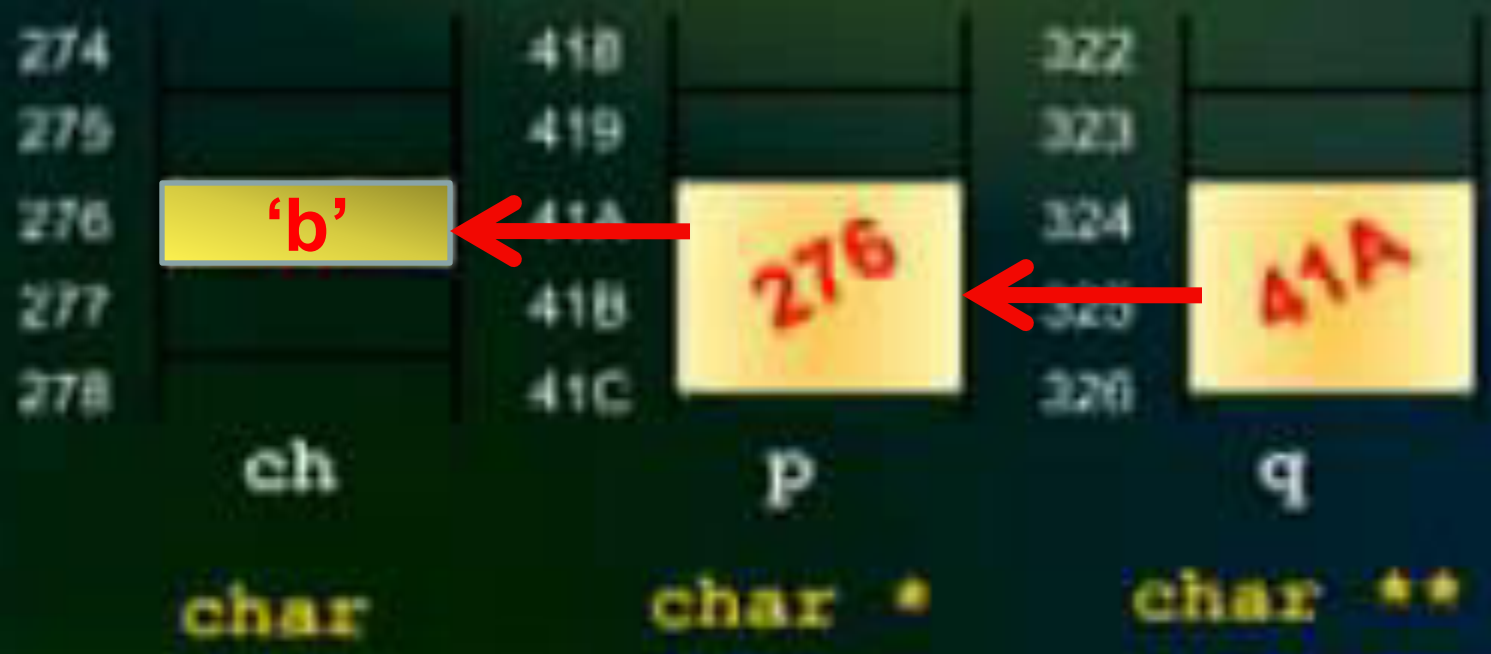
Access the byte No.6 (zero-based) in a `long double` variable

```
long double i,*ptr;

ptr = &i;

*((char *)ptr+6) = 65;
```

Pointer-to-Pointer

```
char ch, *p, **q;

p = &ch;
q = &p;

*p = 'a';

*(*q) = 'b';   OR   **q = 'b';

printf("%c, %c, %c", ch, *p, **q);
```


274
275
276
277
278
'A'
ch
char
418
419
41A
41B
41C
276
p
char *
322
323
324
325
326
41A
q
char **


274
275
276
277
278
'b'
418
419
41A
41B
41C
276
322
323
324
325
326
41A
ch
p
q
char
char *
char **

Points to Ponder

Point 1

- Pointers can also have any other name not only ptr

- For example,
  - int *numb;
  - char *letter;

Point 2

- In Pointer Arithmetic, we can add/subtract any number not only 1

- For example,
  - `numb = numb + 5;`
  - `letter = letter + 10;`

Point 3

In Win32 programming int and long are of same size (4 bytes)

Point 4

- Total number of possible addresses in a 32-bit pointer is 2^{32}.

Lowest address: 0

Highest address: 232-1

=4,29,49,67,295

=0xffffffff

Point 5

RAM addresses are normally written in hexadecimal.

For example

1234 is written as 0x4D2

45564 is written as 0x0000B1FC

RAM addresses are normally written in hexadecimal.

For example

1234 is written as Ox4D2

45564 is written as 0x0000B1FC

Questions

Which bytes are modified?

```
long double i, *ptr;
ptr = &i;
```

```
*((char *) ((long *)ptr + 1)+2) = 91;
*((short int *)((long *)ptr+2)-1)=91;
```