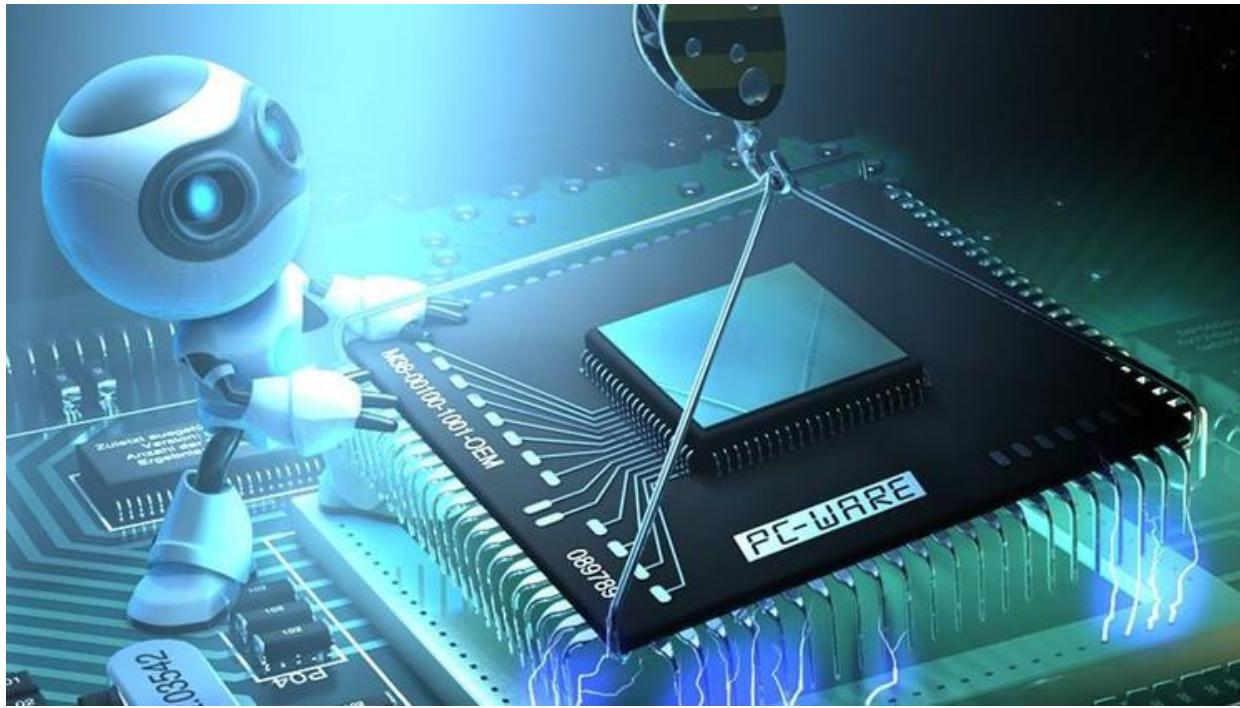


Vector Practise





Structured Data Types

Vectors

- Declaration in C:

```
type name[size];           // indexed starting at 0
```

- Storage in consecutive memory locations

- Access element $V[i]$: $\text{@start } V + i * \text{size}$ (size : size of the elements of V)

V

5	2	-4	1	3
---	---	----	---	---



0

1

2

1

3

$\text{@}V \equiv V[0]$

$V[3] \rightarrow \text{@}V + 3 * 4$

Structured Data Types



Vectors

- Declaration in C:

```
type name[size];           // indexed starting at 0
```

- Storage in consecutive memory locations

- Access element $V[i]$: $\text{@start } V + i * \text{size}$ (size : size of the elements of V)

V

'a'	'b'	'f'	'd'	'k'
0	1	2	3	4

$\text{@}V \equiv V[0]$

$V[3] \rightarrow \text{@}V + 2 * 1$



Structured Data Types

Vectors

- Examples:

Declaration in C	Size of Element	Size of Vector	@element i
char A[12];	1B	12B	@start A + i
char *B[80];	4B	320B	@start B + 4·i
double C[1024];	8B	8KB	@start C + 8·i
int *D[5];	4B	20B	@start D + 4·i
int E[100];	4B	400B	@start E + 4·i

Vectors



Example:

```
int Vi(int v[100], int i) {  
    return v[i];  
}
```

% ecx ← @v

% edx ← i

@v + i * 4

Traduction:

```
Vi: pushl %ebp  
    movl %esp, %ebp  
    movl 8(%ebp), %ecx      # @v → 8[%ebp]  
    movl 12(%ebp), %edx      # i → 12[%ebp]  
    movl (%ecx,%edx,4), %eax  
    popl %ebp                # result in %eax  
    ret
```

$$\% \text{ecx} + \% \text{edx} * 4 \equiv @v + i * 4$$