Addresses and numbers

- Addresses in memory are non-negative binary numbers
  - We can print them using format %p (will print in octal or hexa)

- Although they are non-negative integers, different operations than on ints are possible:
  - Allowed
    • Add or subtract a constant integer from an address
    • Subtract two addresses of same type
  - Not allowed
    • Add, multiply or divide two addresses
    • Multiply or divide 2 addresses by an integer constant,
Pointer arithmetic

First multiply sizeof (type) * increment

<table>
<thead>
<tr>
<th>type</th>
<th>sizeof(type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
</tr>
</tbody>
</table>

Pointer points to address $p:1000_{10}$

<table>
<thead>
<tr>
<th>Type</th>
<th>$p + 3$</th>
<th>$p - 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>double * p</td>
<td>$1000 + 3*8 \rightarrow 1024$</td>
<td>$1000 - 2*8 \rightarrow 984$</td>
</tr>
<tr>
<td>long * p</td>
<td>$1000 + 3*4 \rightarrow 1012$</td>
<td>$1000 - 2*4 \rightarrow 992$</td>
</tr>
<tr>
<td>short * p</td>
<td>$1000 + 3*2 \rightarrow 1006$</td>
<td>$1000 - 2*2 \rightarrow 996$</td>
</tr>
<tr>
<td>char * p</td>
<td>$1000 + 3*1 \rightarrow 1003$</td>
<td>$1000 - 2*1 \rightarrow 998$</td>
</tr>
<tr>
<td>void * p</td>
<td>$1000 + 3 \rightarrow 1003$</td>
<td>$1000 - 2 \rightarrow 998$</td>
</tr>
</tbody>
</table>
### Pointer arithmetic

<table>
<thead>
<tr>
<th>type</th>
<th>sizeof(type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
</tr>
</tbody>
</table>

- $p = 1000_{10}$, $q = 1024_{10}$
- $q$ is 24 bytes ahead of $p$

<table>
<thead>
<tr>
<th>type</th>
<th>$q - p$</th>
<th>$p - q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>double * p, * q</td>
<td>$\frac{1024 - 1000}{8} \to 3$</td>
<td>$-3$</td>
</tr>
<tr>
<td>long * p, * q</td>
<td>$\frac{1024 - 1000}{4} \to 6$</td>
<td>$-6$</td>
</tr>
<tr>
<td>short * p, * q</td>
<td>$\frac{1024 - 1000}{2} \to 12$</td>
<td>$-12$</td>
</tr>
<tr>
<td>char * p, * q</td>
<td>$\frac{1024 - 1000}{1} \to 24$</td>
<td>$-24$</td>
</tr>
<tr>
<td>void * p, * q</td>
<td>$1024 - 1000$</td>
<td>$-24$</td>
</tr>
</tbody>
</table>

The type that is always an int is `ptrdiff_t`. The typical header is `stddef.h`. The type is used in the C standard, because it is always the integer type of the largest integer type (in general). This means that `long int` and `int` are always the same type (or a superset of it).
Operators in pointer arithmetic

- I is an int
- P, q are pointer (of same type).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
<th>Validity</th>
<th>Type</th>
<th>Validity</th>
</tr>
</thead>
<tbody>
<tr>
<td>p + i</td>
<td>int</td>
<td>permitted</td>
<td>i + p</td>
<td>permitted</td>
</tr>
<tr>
<td>p - i</td>
<td>int</td>
<td>forbidden</td>
<td>i - p</td>
<td>forbidden</td>
</tr>
<tr>
<td>p = p + i</td>
<td>int</td>
<td>permitted</td>
<td>p += i</td>
<td>forbidden</td>
</tr>
<tr>
<td>p = p - i</td>
<td>int</td>
<td>permitted</td>
<td>p -= i</td>
<td>forbidden</td>
</tr>
<tr>
<td>p += 1</td>
<td>int</td>
<td>permitted</td>
<td>++ p</td>
<td>permitted</td>
</tr>
<tr>
<td>p += 1</td>
<td>int</td>
<td>permitted</td>
<td>p ++</td>
<td>permitted</td>
</tr>
<tr>
<td>p -= 1</td>
<td>int</td>
<td>permitted</td>
<td>-- p</td>
<td>permitted</td>
</tr>
<tr>
<td>p -= 1</td>
<td>int</td>
<td>permitted</td>
<td>p --</td>
<td>permitted</td>
</tr>
<tr>
<td>p + q</td>
<td>ptrdiff_t</td>
<td>permitted</td>
<td>p - q</td>
<td>permitted</td>
</tr>
<tr>
<td>p += q</td>
<td>ptrdiff_t</td>
<td>permitted</td>
<td>p -= q</td>
<td>permitted</td>
</tr>
<tr>
<td>p -= q</td>
<td>ptrdiff_t</td>
<td>permitted</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays and pointers

- Array name – address of first element of array.
- Pointer arithmetic is possible on arrays

```c
double salaries[3];

&salaries[0]  salaries + 0
&salaries[1]  salaries + 1
&salaries[2]  salaries + 2
&salaries[i]  salaries + i

salaries[i]  def  *(salaries + i)

&salaries[i]  &*(salaries + i)  salaries + i
```
Size of array (sizeof())

- sizeof() will return the number of bytes taken by a variable:
  ```c
  double array[5];
  sizeof(array) == 5 * sizeof(double) == 40 •
  *array == array[0] therefore:
  sizeof(*array) == sizeof(double) == 8
  ```

However:

```c
sizeof(array) == 5 * sizeof(*array)
```

<table>
<thead>
<tr>
<th>sizeof(a) / sizeof(*a)</th>
<th>Number of</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pointers to array elements

Remember: if p is the address of a variable, *p will give its contents.

Likewise if p points to an element of the array, the element can be obtained by *p.

p+i is the address of an element - i places forward from p.

p-i is the address of an element - i places backwards from p.

p[i] == *(p+i) is the element - I places forward.

p[-i] == *(p-i) is the element - I places backward from p.

All this if true if:

- a is an array of type T
- P is a pointer to type T
- And there has been an assignment p = a
Arrays and pointers – memory

double salaries[3];
double *p = salaries;

++p;
p = &salaries[2];
printf("%g ", *p);
printf("%g ", *salaries);
if ( p == salaries+2 )
    printf("%g\n", *(p-1));

25.1 3.5 19.4
Out of bounds

- In arrays valid indices are 0- K–1.
- In C, there is no checking of those bounds at compilation time.
- At runtime, several possibilities exist:
  - The program crashes (immediately or not)
  - The program does not crash, returns wrong results.
- However with pointers some possibilities which look wrong are working:
  - P[i] for i=-6,…6 will be valid
  - P[9] won’t be valid.

```c
int a[13];
int *p = a+6;  /* p points to a[6] */
```
Pointers to strings

```c
char name[] = "My name is Luka";

char *p;
char *q = "My name is Luka";
char *r = name;

p = "Just don't argue any more!";
r = p;
p = "http://www.youtube.com/watch?v=RZyxYL753w4";
name = p;       /* SYNTAX ERROR !!! */

printf("%d %d\n", strlen(p), strlen("Suzanne Vega"));
```
**strcmp**

- comparison of 2 strings in lexicographic order
- If the left string is “smaller”, a negative number is returned
- If the right string is “smaller”, a positive number is returned
- Otherwise 0 is returned

```c
int strcmp(char * left, char * right)
{
    for ( ; *left && *right; left++, right++ )
        if ( *left != *right )
            break;
    return (unsigned char)*left - (unsigned char)*right;
}
```