# Pointers – Strings

## Basics of Programming 1


G. Horváth, A.B. Nagy, Z. Zsóka, P. Fiala, A. Vitéz

16 October, 2024

# Content

Chapter 1

# Pointers

Fundamental Theorem of Software Engineering (FTSE)

*"We can solve any problem*
*by introducing an extra level of indirection."*
Andrew Koenig

# Where are the variables?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program that lists the address and value of variables

```
1 int a = 2;
2 double b = 8.0;
3 printf("address of a: %p, its value: %d\n", &a, a);
4 printf("address of b: %p, its value: %f\n", &b, b);
```

```
address of a:  0x7fffa3a4225c, its value:  2
address of b:  0x7fffa3a42250, its value:  8.000000
```

[1] more precisely left-values

# Where are the variables?

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

Let's write a program that lists the address and value of variables

```
1 int a = 2;
2 double b = 8.0;
3 printf("address of a: %p, its value: %d\n", &a, a);
4 printf("address of b: %p, its value: %f\n", &b, b);
```

```
address of a:  0x7fffa3a4225c, its value:  2
address of b:  0x7fffa3a42250, its value:  8.000000
```

- address of variable: starting address of "memory block" containing the variable, expressed in bytes
- with the address-of operator we can create address of any variables[1] like this `&<reference>`

[1] more precisely left-values

# The pointer type

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

The pointer type is for storing memory addresses

## Declaration of pointer

```
<pointed type> * <identifier>;
```

```
1  int*    p;  /*   p stores the address of one int data */
2  double* q;  /*   q stores the address of one double data */
3  char*   r;  /*   r stores the address of one char data */
```

# The pointer type

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

The pointer type is for storing memory addresses

## Declaration of pointer

```
<pointed type> * <identifier>;
```

```
1  int*    p;  /*   p stores the address of one int data */
2  double* q;  /*   q stores the address of one double data */
3  char*   r;  /*   r stores the address of one char data */
```

it is the same, even if arranged in a different way

```
1  int    *p;  /*   p stores the address of one int data */
2  double *q;  /*   q stores the address of one double data */
3  char   *r;  /*   r stores the address of one char data */
```

# Operator of indirection



- If pointer p stores the address of variable a, then p "points to a"

# Operator of indirection



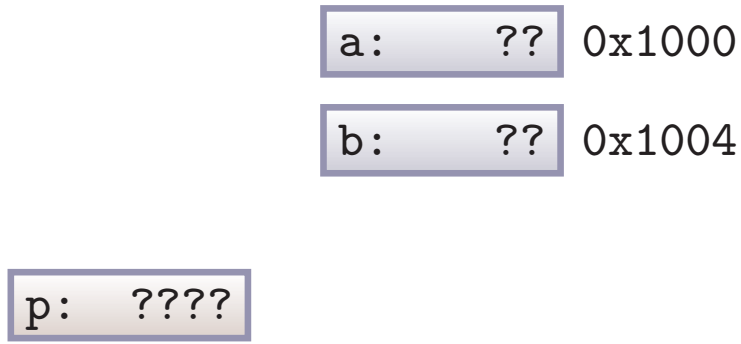
- If pointer `p` stores the address of variable `a`, then `p` "points to `a`"
- If `p` points to `a`, then variable `a` can be accessed as `*p`. Here `*` is the operator of indirection (dereference operator).

# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

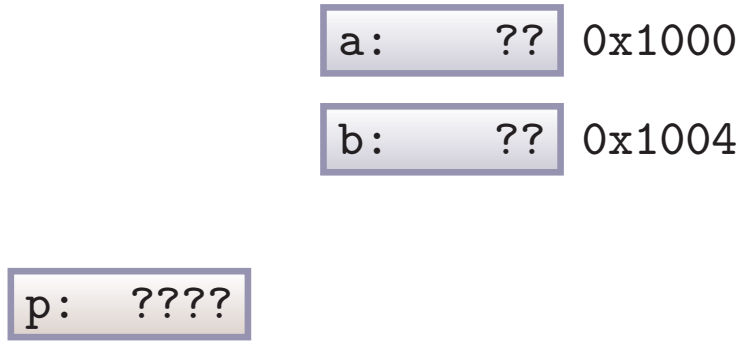


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

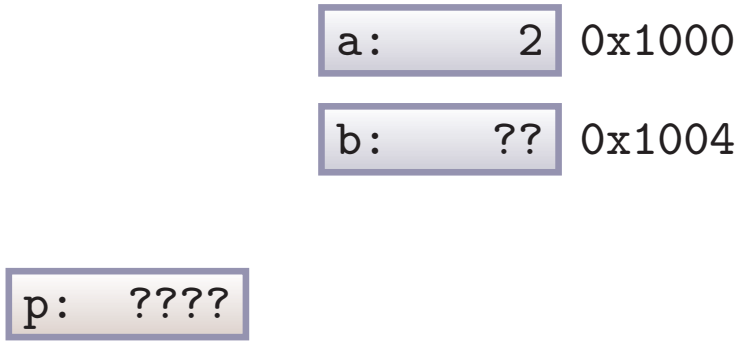


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

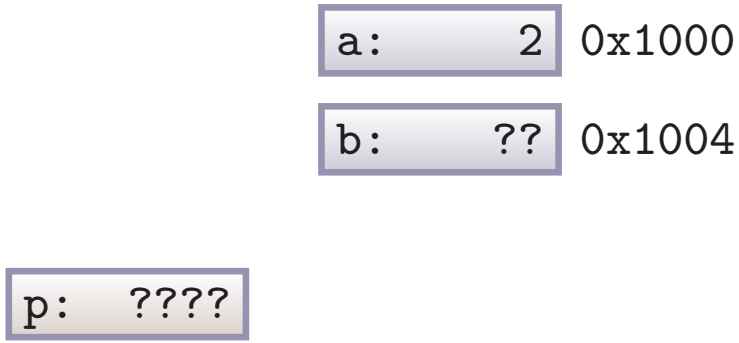


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

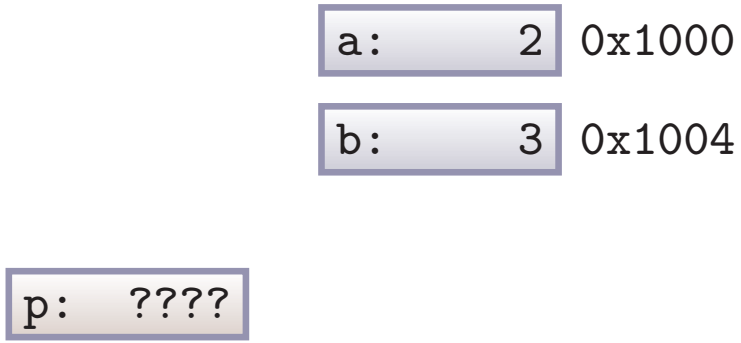


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

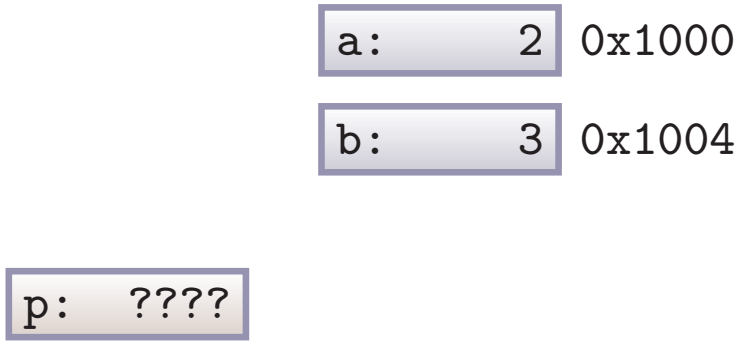


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

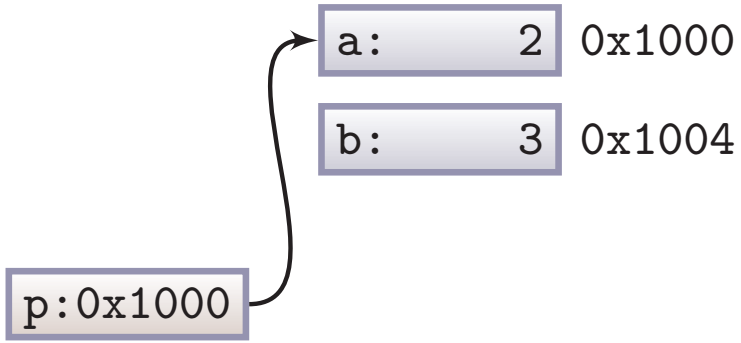


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

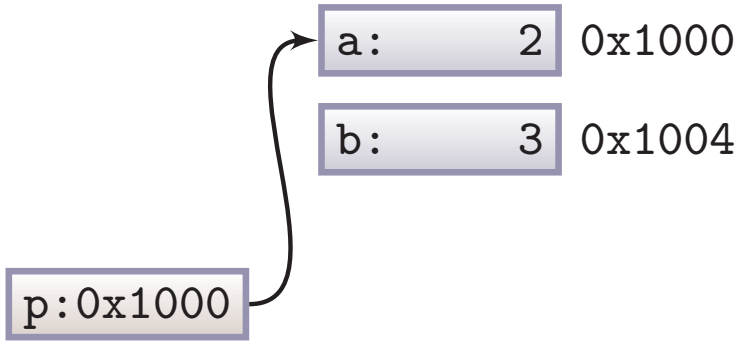


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

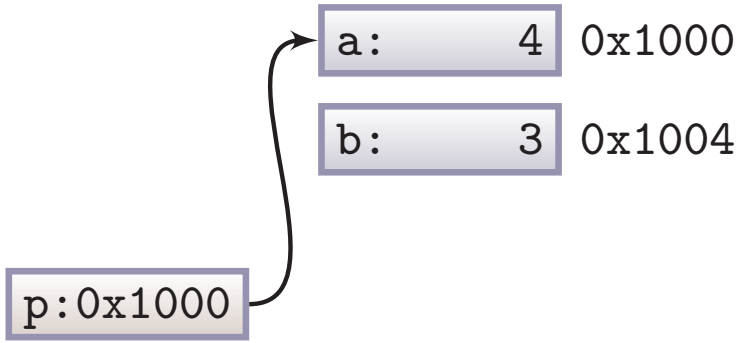


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

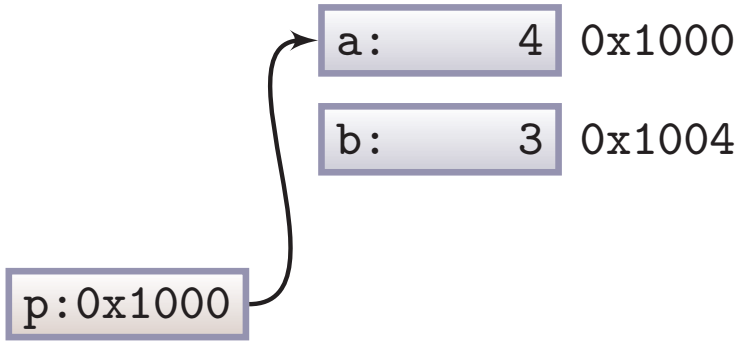


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

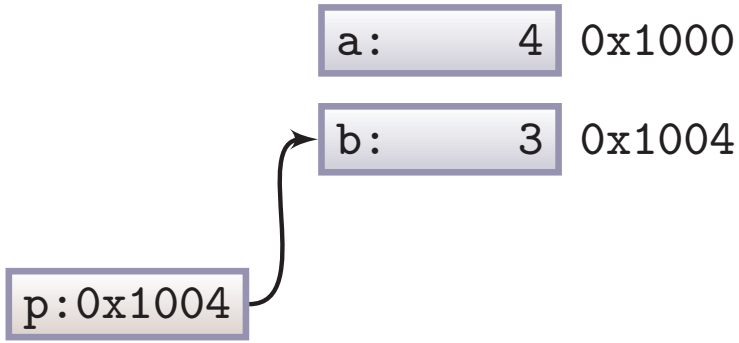


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

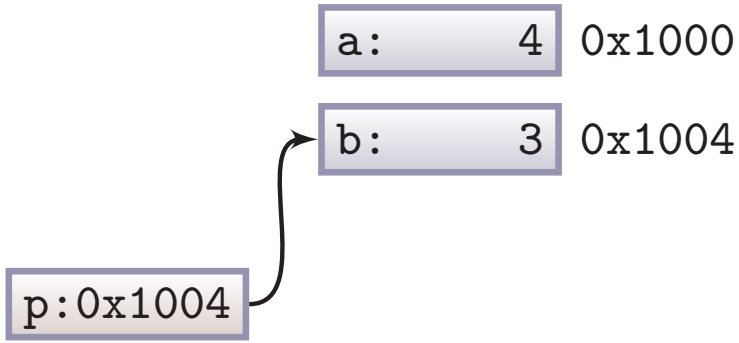


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```

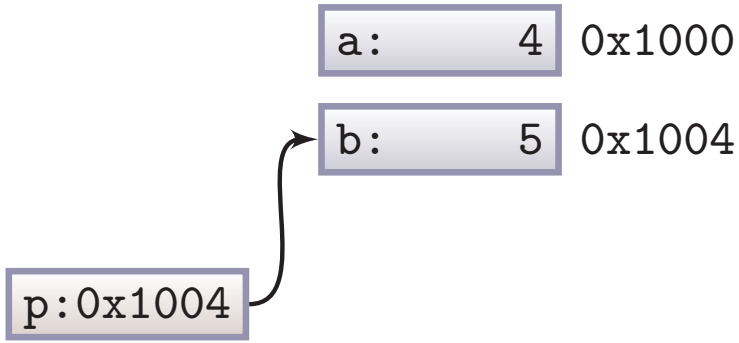


# Operator of indirection

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If pointer p stores the address of variable a, then p "points to a"
- If p points to a, then variable a can be accessed as *p. Here * is the operator of indirection (dereference operator).

```
1  int a, b;
2  int *p; /* int pointer */
3
4  a = 2;
5  b = 3;
6  p = &a; /* p points to a */
7  *p = 4; /* a = 4 */
8  p = &b; /* p points to b */
9  *p = 5; /* b = 5 */
```



# Address-of and indirection – summary



| operator | operation | description |
|---|---|---|
| & | address-of | assigns its address to the variable |
| * | indirection | assigns variable to the address |

# Address-of and indirection – summary



| operator | operation | description |
|---|---|---|
| & | address-of | assigns its address to the variable |
| * | indirection | assigns variable to the address |

- Interpreting declaration: type of *p is `int`

```
1  int *p;        /* get used to this version */
```

# Address-of and indirection – summary



| operator | operation | description |
|---|---|---|
| & | address-of | assigns its address to the variable |
| * | indirection | assigns variable to the address |

- Interpreting declaration: type of *p is `int`

```
1 int *p;       /* get used to this version */
```

- Multiple declaration: type of a, *p and *q is `int`

```
1 int a, *p, *q; /* at least because of this */
```

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

# Application – Function for exchanging two variables

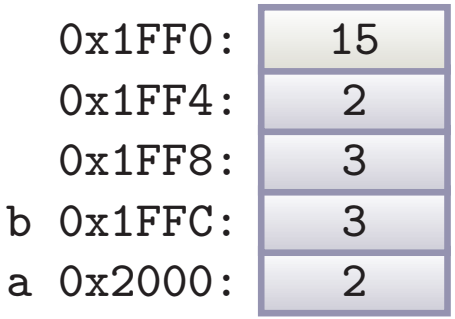
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

| | | |
|---|---|---|
| | 0x1FF0: | 15 |
| x | 0x1FF4: | 2 |
| y | 0x1FF8: | 3 |
| b | 0x1FFC: | 3 |
| a | 0x2000: | 2 |

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

| | | |
|---|---|---|
| tmp | 0x1FEC: | 2 |
| | 0x1FF0: | 15 |
| x | 0x1FF4: | 2 |
| y | 0x1FF8: | 3 |
| b | 0x1FFC: | 3 |
| a | 0x2000: | 2 |

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

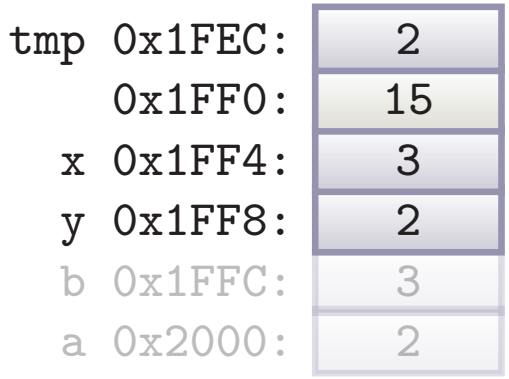
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

| | | |
|---|---|---|
| tmp | 0x1FEC: | 2 |
| | 0x1FF0: | 15 |
| x | 0x1FF4: | 3 |
| y | 0x1FF8: | 3 |
| b | 0x1FFC: | 3 |
| a | 0x2000: | 2 |

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

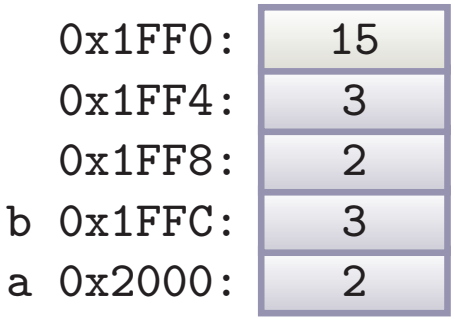
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

| | | |
|---|---|---|
| | 0x1FF0: | 15 |
| x | 0x1FF4: | 3 |
| y | 0x1FF8: | 2 |
| b | 0x1FFC: | 3 |
| a | 0x2000: | 2 |

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

# Application – Function for exchanging two variables

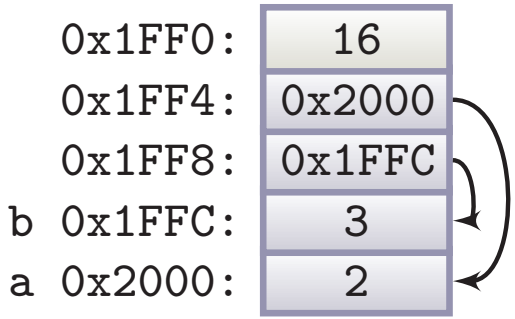
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

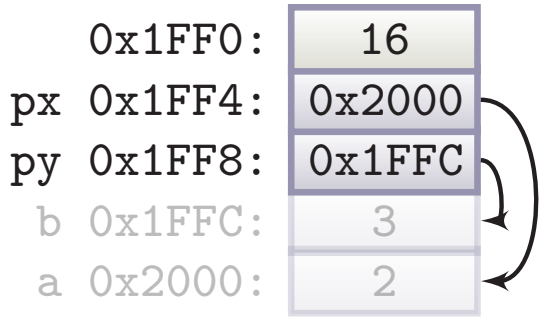
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

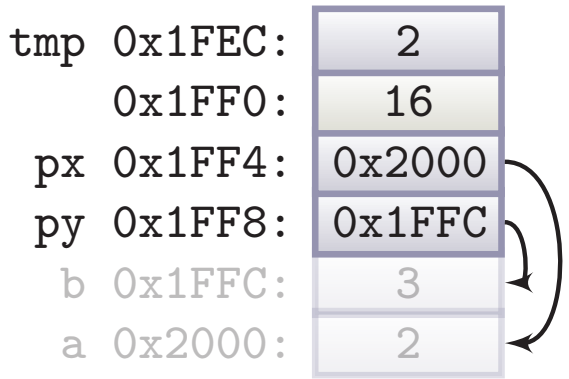
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

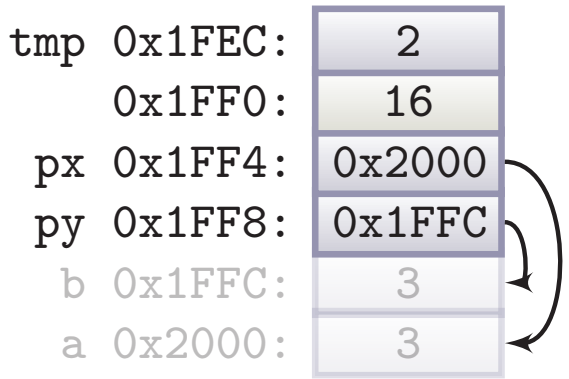
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

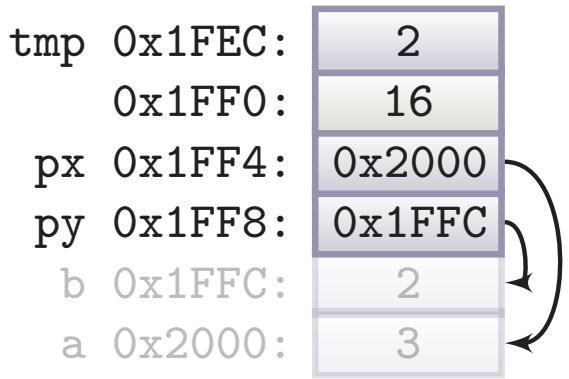
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

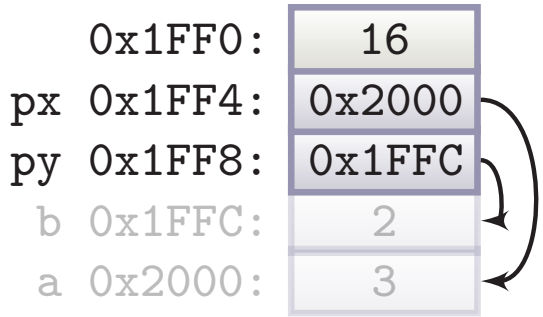
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

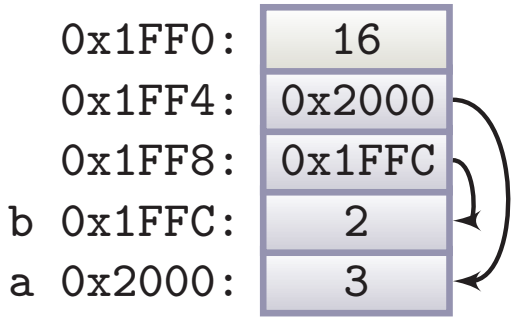
```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```



# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

# Application – Function for exchanging two variables

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

```
1  void xchg(int x, int y) {
2    int tmp = x;
3    x = y;
4    y = tmp;
5  }
6
7  void xchgp(int *px, int *py) {
8    int tmp = *px;
9    *px = *py;
10   *py = tmp;
11 }
12
13 int main(void) {
14   int a = 2, b = 3;
15   xchg(a, b);
   /* NO exchange */
16   xchgp(&a, &b);/* exchange */
17   return 0;
18 }
```

# Application – returning value as parameter



- If a function has to calculate several values, then. . .

  . . . we can use structures, but sometimes this seems rather unnecessary.

# Application – returning value as parameter

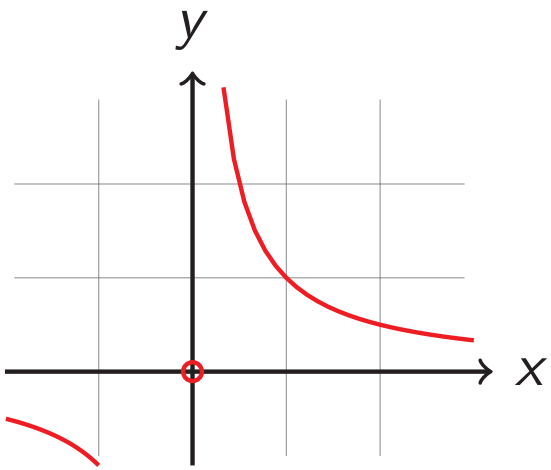
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If a function has to calculate several values, then...

  ...we can use structures, but sometimes this seems rather unnecessary.
  Instead...

```
1  int inverse(double x, double *py)
2  {
3    if (abs(x) < 1e-10) return 0;
4    *py = 1.0 / x;
5    return 1;
6  }
```

link



# Application – returning value as parameter

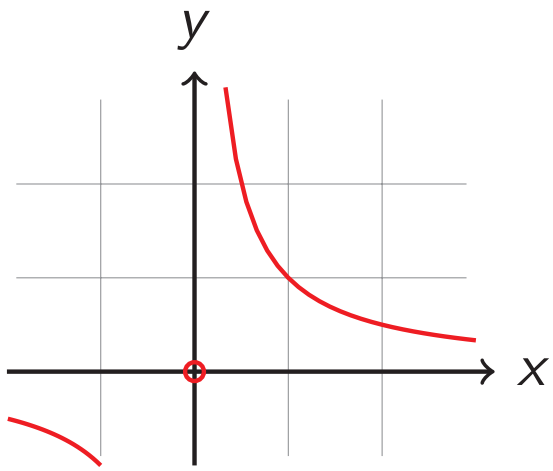
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- If a function has to calculate several values, then...

  ...we can use structures, but sometimes this seems rather unnecessary.
  Instead...

```c
int inverse(double x, double *py)
{
  if (abs(x) < 1e-10) return 0;
  *py = 1.0 / x;
  return 1;
}
```

link




```c
double y;          /* memory allocation for result */
if (inverse(5.0, &y) == 1)
  printf("Reciprocal of %f is %f\n", 5.0, y);
else
  printf("Reciprocal does not exist");
```

link

# Application – return values as parameters



- Now we understand what this means

```
1 int n, p;
2 /* return value as parameter */
3 scanf("%d%d", &n, &p); /* we pass the addresses */
```

# Remarks:



- What is the use of having different pointer types for different types?

# Remarks:



- What is the use of having different pointer types for different types?
- Type = set of values + operations
- Obviously set of values is the same for all pointers (unsigned integer addresses)
- Operations are different!

# Remarks:



- What is the use of having different pointer types for different types?
- Type = set of values + operations
- Obviously set of values is the same for all pointers (unsigned integer addresses)
- Operations are different!
- The operator of indirection (*)
  - makes `int` from `int` pointer
  - makes `char` from `char` pointer

# Remarks:



- What is the use of having different pointer types for different types?
- Type = set of values + operations
- Obviously set of values is the same for all pointers (unsigned integer addresses)
- Operations are different!
- The operator of indirection (*)
  - makes `int` from `int` pointer
  - makes `char` from `char` pointer
- Other differences are detailed in pointer-arithmetics. . .

# Pointer-arithmetics



If `p` and `q` are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| `p+1` | pointer | points to the next element |
| `p-1` | pointer | points to the previous element |
| `q-p` | integer number | number of elements between two addresses |

[2] In this example we assume that size of `int` is 4 bytes

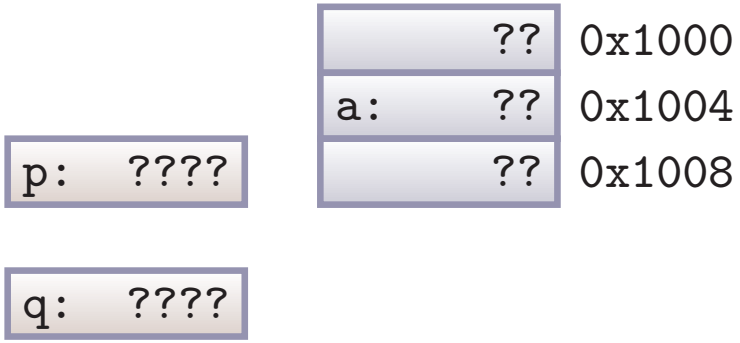
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

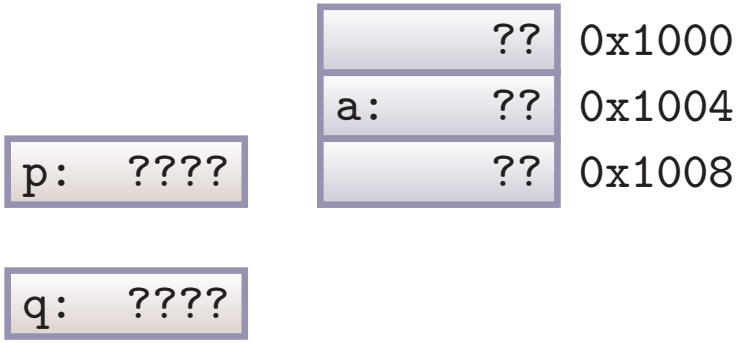
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

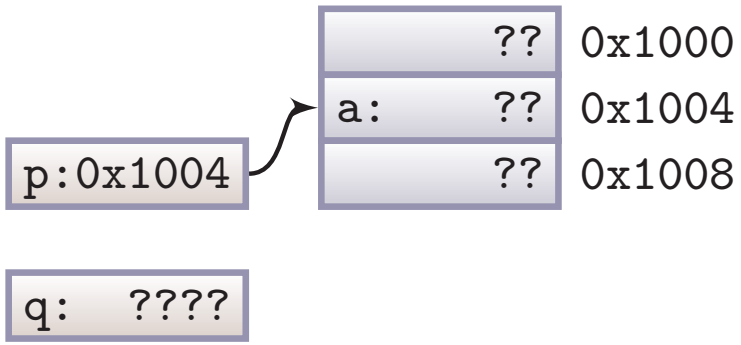
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

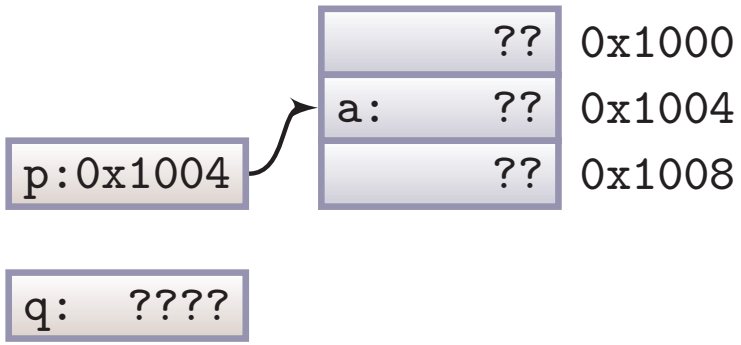
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

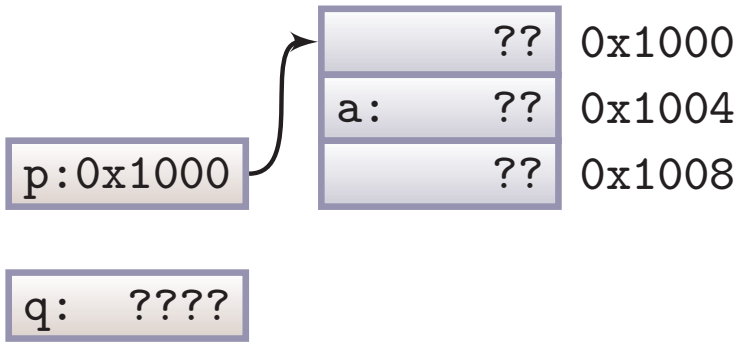
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

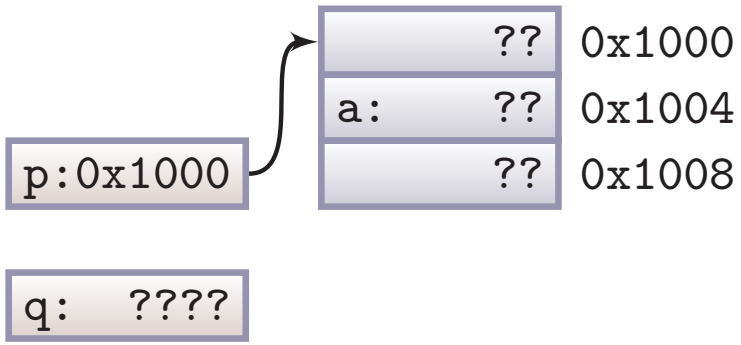
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

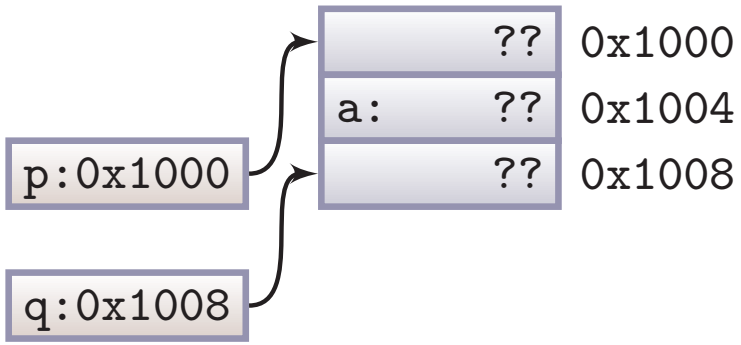
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

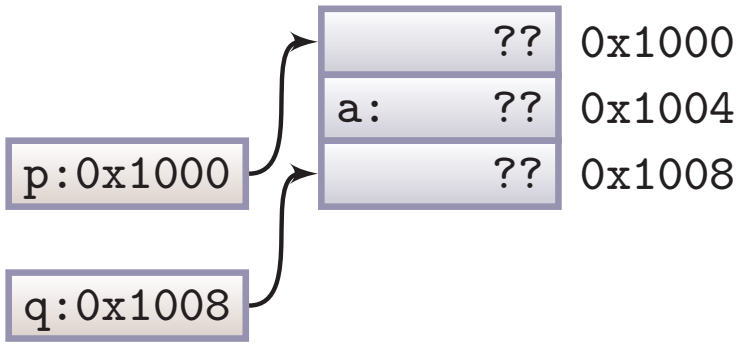
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




[2] In this example we assume that size of `int` is 4 bytes

# Pointer-arithmetics

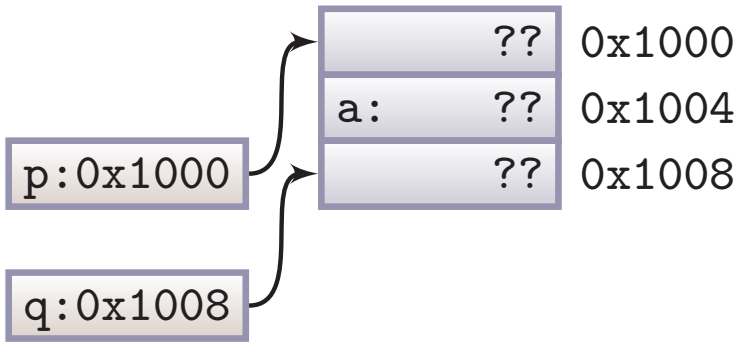
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```

```
2
```




[2] In this example we assume that size of `int` is 4 bytes

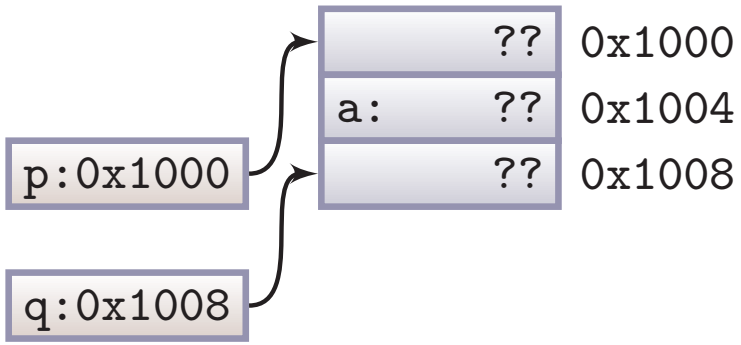
# Pointer-arithmetics

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

If p and q are pointers of the same type, then

| expr. | type | meaning |
|---|---|---|
| p+1 | pointer | points to the next element |
| p-1 | pointer | points to the previous element |
| q-p | integer number | number of elements between two addresses |

```
1  int a, *p, *q;
2
3  p = &a;
4  p = p-1;
5  q = p+2;
6  printf("%d", q-p);
```




**2**

- At pointer-arithmetic operaitons addresses are "measured" in the representation size of the pointed type, and not in bytes.[2]

[2] In this example we assume that size of `int` is 4 bytes

# Pointer-arithmetic



- In the above example pointer-arithmetic is strange, as we don't know what is before or after variable a in the memory.
- This operation is meaningful, when we have variables of the same type, stored in the memory one afte the other.
- This is the case for arrays.

# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Traversing an array can be done with pointer-arithmetics.

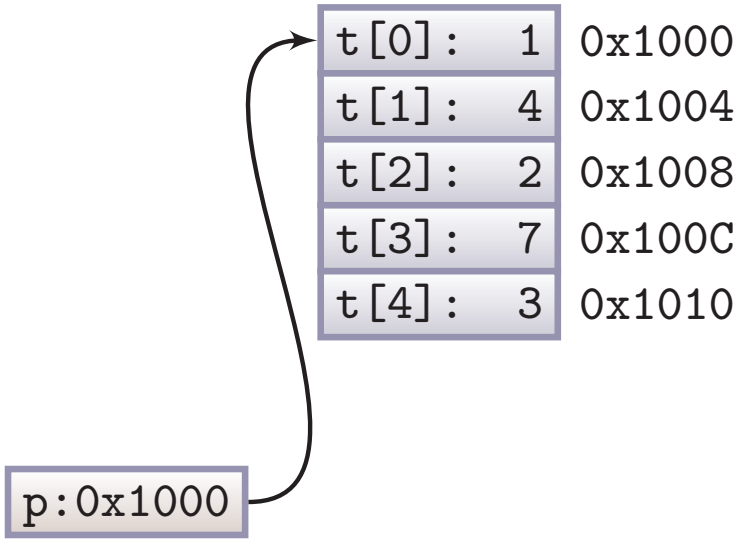
# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Traversing an array can be done with pointer-arithmetics.

```
1  int t[5] = {1,4,2,7,3};
2  int *p, i;
3
4  p = &t[0];
5  for (i = 0; i < 5; ++i)
6    printf("%d ", *(p+i));
```

```
1 4 2 7 3
```



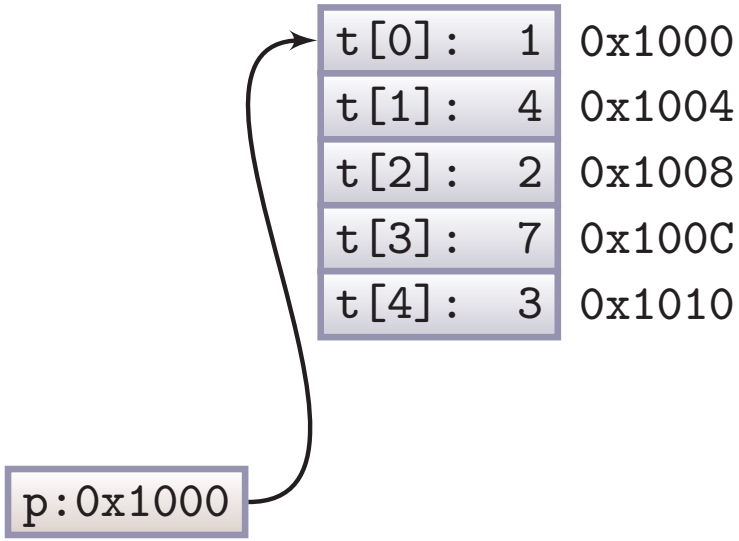
# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Traversing an array can be done with pointer-arithmetics.

```
1  int t[5] = {1,4,2,7,3};
2  int *p, i;
3
4  p = &t[0];
5  for (i = 0; i < 5; ++i)
6    printf("%d ", *(p+i));
```

```
1 4 2 7 3
```




- In this example `*(p+i)` is the same as `t[i]`, because `p` points to the beginning of array `t`

# Pointers and arrays



- Pointers can be taken as arrays, this means they can be indexed.
  By definition `p[i]` is identical to `*(p+i)`

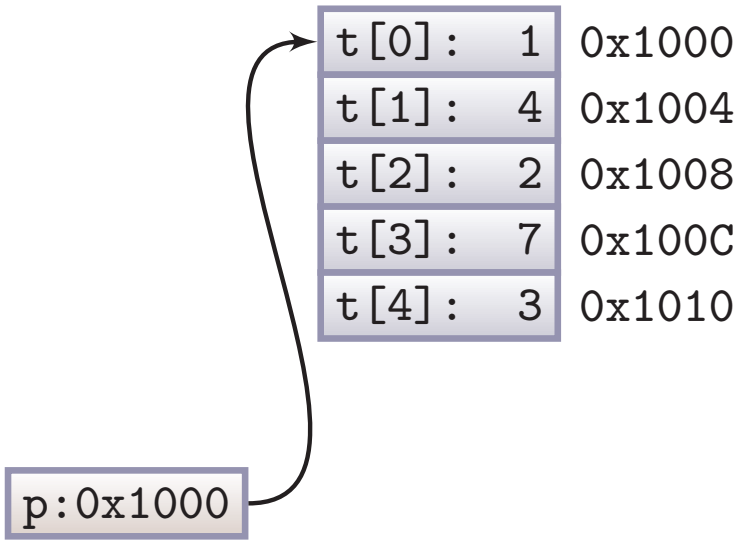
# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pointers can be taken as arrays, this means they can be indexed.
  By definition `p[i]` is identical to `*(p+i)`

```
1  int t[5] = {1,4,2,7,3};
2  int *p, i;
3
4  p = &t[0];
5  for (i = 0; i < 5; ++i)
6    printf("%d ", p[i]);
```

```
1 4 2 7 3
```



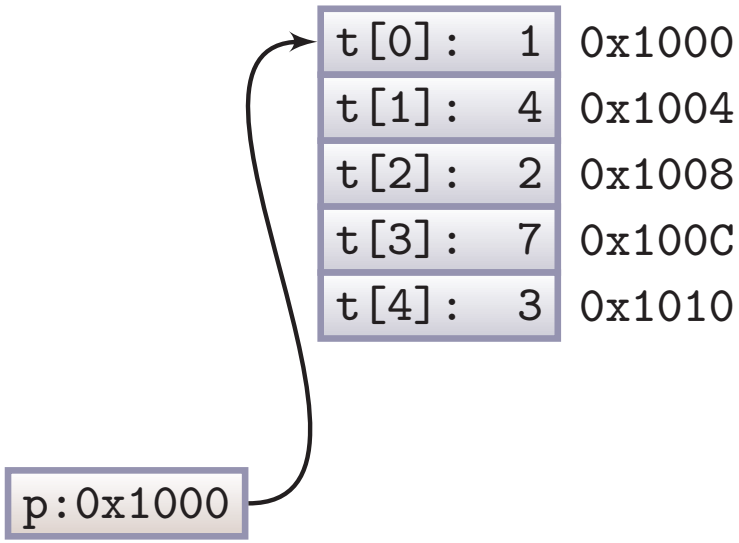
# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pointers can be taken as arrays, this means they can be indexed.
  By definition `p[i]` is identical to `*(p+i)`

```
1  int t[5] = {1,4,2,7,3};
2  int *p, i;
3
4  p = &t[0];
5  for (i = 0; i < 5; ++i)
6    printf("%d ", p[i]);
```

```
1 4 2 7 3
```




- In this example `p[i]` is the same as `t[i]`, because `p` points to the beginning of array `t`

# Pointers and arrays



- Arrays can be taken as pointers.
  The identifier (name) of array is the starting address of the array, in other words the value of expression `t` is `&t[0]`

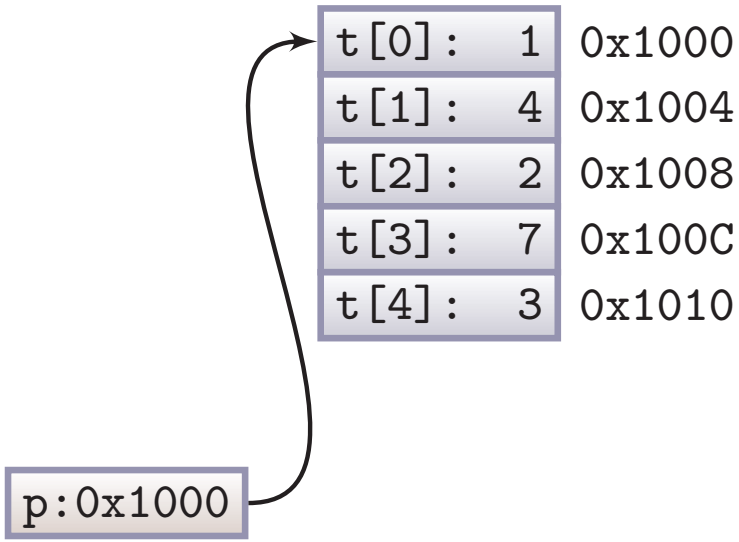
# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Arrays can be taken as pointers.
  The identifier (name) of array is the starting address of the array, in other words the value of expression `t` is `&t[0]`

```
1  int t[5] = {1,4,2,7,3};
2  int *p, i;
3
4  p = t; /* &t[0] */
5  for (i = 0; i < 5; ++i)
6    printf("%d ", p[i]);
```

```
1 4 2 7 3
```



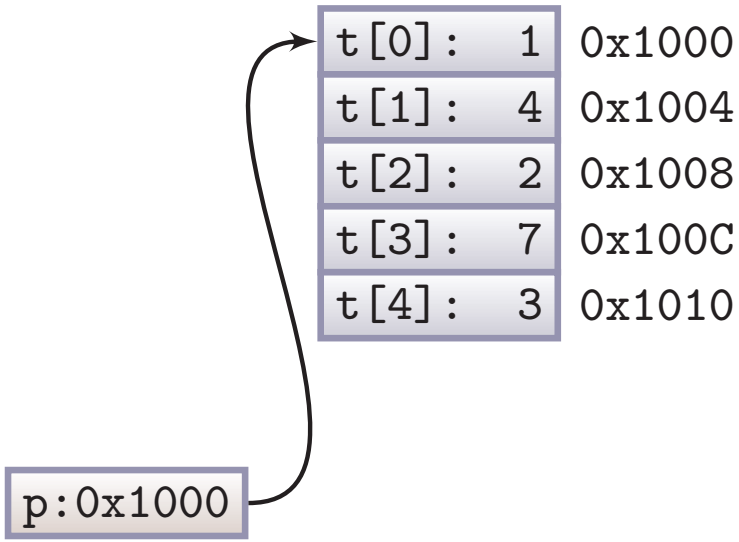
# Pointers and arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Arrays can be taken as pointers.
  The identifier (name) of array is the starting address of the array, in other words the value of expression `t` is `&t[0]`

```
1  int t[5] = {1,4,2,7,3};
2  int *p, i;
3
4  p = t; /* &t[0] */
5  for (i = 0; i < 5; ++i)
6    printf("%d ", p[i]);
```

```
1 4 2 7 3
```




- Pointer-arithmetics work for arrays too:
  `t+i` is identical to `&t[i]`

# Pointers and arrays – summary



- Pointer can be taken as array, and array as a pointer.

# Pointers and arrays – summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pointer can be taken as array, and array as a pointer.
- index operator is only a notation
  the compiler will always replace `a[i]` with `*(a+i)`,
  both if *a* is pointer, and also if *a* is array.

# Pointers and arrays – summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pointer can be taken as array, and array as a pointer.
- index operator is only a notation
  the compiler will always replace `a[i]` with `*(a+i)`,
  both if *a* is pointer, and also if *a* is array.
- Differences:
  - Elements of array have allocated space in memory (variables).
    No allocated elements belong to the pointer.

# Pointers and arrays – summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pointer can be taken as array, and array as a pointer.
- index operator is only a notation
  the compiler will always replace `a[i]` with `*(a+i)`,
  both if *a* is pointer, and also if *a* is array.
- Differences:
  - Elements of array have allocated space in memory (variables).
    No allocated elements belong to the pointer.
  - Starting address of array is constant, it cannot be changed.
    Pointer is a variable, the address stored in it can be modified.

# Pointers and arrays – summary

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Pointer can be taken as array, and array as a pointer.
- index operator is only a notation
  the compiler will always replace `a[i]` with `*(a+i)`,
  both if *a* is pointer, and also if *a* is array.
- Differences:
  - Elements of array have allocated space in memory (variables).
    No allocated elements belong to the pointer.
  - Starting address of array is constant, it cannot be changed.
    Pointer is a variable, the address stored in it can be modified.

```
1  int array[5] = {1, 3, 2, 4, 7};
2  int *p = array;
3
4  /* the elements can be accessed via p and a */
5  p[0] = 2;              array[0] = 2;
6  *p = 2;                *array = 2;
7
8  /* p can be changed   array CANNOT */
9  p = p+1; /* ok */      array = array + 1; /* ERROR */
```

# Passing arrays to functions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's use a function to determine the first negative element of `array`!

[3] defined in `stdio.h`

# Passing arrays to functions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's use a function to determine the first negative element of `array`!
- Passing an array:
  - Address of first element `double*`
  - Size of the array `typedef unsigned int size_t`[3]

[3] defined in `stdio.h`

# Passing arrays to functions



- Let's use a function to determine the first negative element of array!
- Passing an array:
  - Address of first element `double*`
  - Size of the array `typedef unsigned int size_t`[3]

```c
double first_negative(double *array, size_t size)
{
  size_t i;
  for (i = 0; i < size; ++i)  /* for each elems. */
    if (array[i] < 0.0)
      return array[i];

  return 0; /* all are non-negative */
}
```

link

```c
double myarray[3] = {3.0, 1.0, -2.0};
double neg = first_negative(myarray, 3);
```

link

[3] defined in `stdio.h`

# Passing arrays to functions



- To distinguish arrays and pointers in the parameter list, we can use the array-notation when passing an array.

```
1  double first_negative(double array[], size_t size)
2                  /*   (double *array,  size_t size)  */
3  {
4     ...
5  }
```

# Passing arrays to functions



- To distinguish arrays and pointers in the parameter list, we can use the array-notation when passing an array.

```
1 double first_negative(double array[], size_t size)
2                  /*   (double *array,  size_t size)  */
3 {
4    ...
5 }
```

- In the formal parameter list `double a[]` is identical to `double *a`.
- In the formal parameter list we can use only empty `[]`, and size should be passed as a separate parameter!

# Passing arrays to functions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's use a function to determine the first negative element of array!
- The return value should be the address of the element found.

```
1  double *first_negative(double *array, size_t size)
2  {
3    size_t i;
4    for (i = 0; i < size; ++i)  /* for each elems. */
5      if (array[i] < 0.0)
6        return &array[i];
7
8    return NULL; /* all are non-negative */
9  }
```

link

# Passing arrays to functions

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Let's use a function to determine the first negative element of array!
- The return value should be the address of the element found.

```
1  double *first_negative(double *array, size_t size)
2  {
3    size_t i;
4    for (i = 0; i < size; ++i)  /* for each elems. */
5      if (array[i] < 0.0)
6        return &array[i];
7
8    return NULL; /* all are non-negative */
9  }
```

link

# Null pointer

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The null pointer (`NULL`)

# Null pointer

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- The null pointer (`NULL`)
  - It stores the `0x0000` address

# Null pointer



- The null pointer (`NULL`)
  - It stores the `0x0000` address
  - Agreed that it "points to nowhere"

Chapter 2

# Strings

# Strings

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- In C, text is stored in character arrays with termination sign, called as strings.

# Strings



- In C, text is stored in character arrays with termination sign, called as strings.
- The termination sign is the character with 0 ASCII-code `'\0'`, the null-character.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 'S' | 'o' | 'm' | 'e' | ' ' | 't' | 'e' | 'x' | 't' | '\0' |

# Defining strings as character arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Definition of character array with initialization

```
1  char s[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

# Defining strings as character arrays

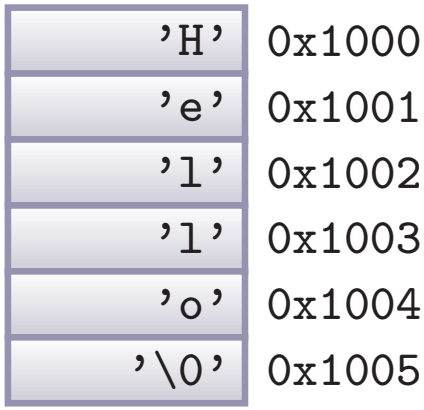
DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Definition of character array with initialization

```
1 char s[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

- The same in a more simple way

```
1 char s[] = "Hello"; /* s array (const.addr 0x1000) */
```



# Defining strings as character arrays

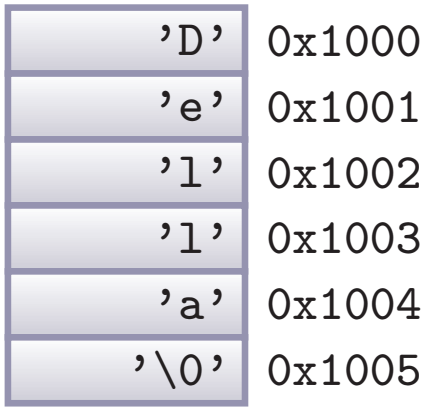


- Definition of character array with initialization

```
1 char s[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

- The same in a more simple way

```
1 char s[] = "Hello"; /* s array (const.addr 0x1000) */
```




- Elements of s can be accessed with indexing or with pointer-arithmetics

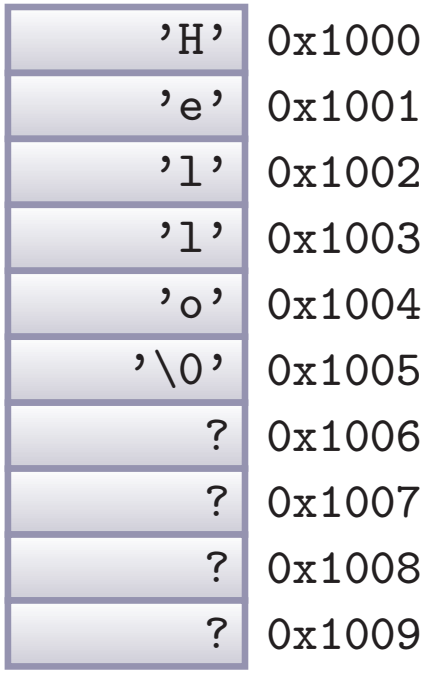
```
1 *s = 'D';    /* s is taken as pointer */
2 s[4] = 'a'; /* s is taken as array */
```

# Defining strings as character arrays



- We can allocate memory for a longer string than needed now, thus we have an overhead.

```
1  char s[10] = "Hello"; /* s array, (const.addr. 0x1000) */
```

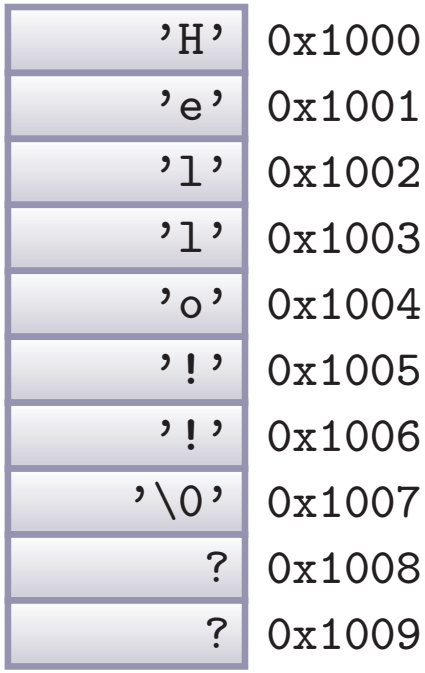


# Defining strings as character arrays

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- We can allocate memory for a longer string than needed now, thus we have an overhead.

```c
char s[10] = "Hello"; /* s array, (const.addr. 0x1000) */
```




- Modification:

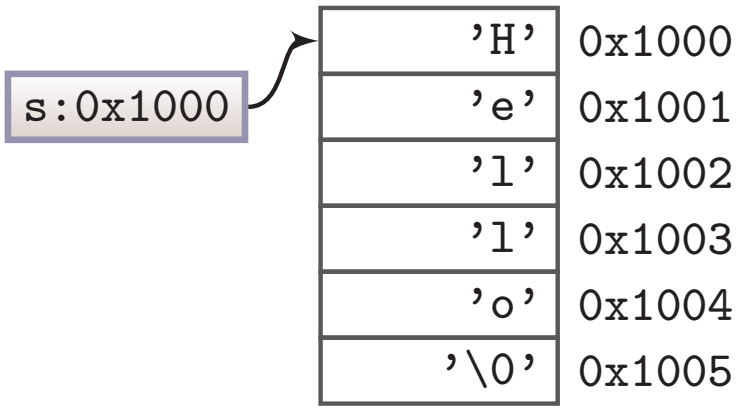
```c
s[5] = s[6] = '!';
s[7] = '\0';          /* must be terminated */
```

# Defining strings as character arrays



- Defining a constant character array and a pointer pointing to it, with initialization.

```
1  char *s = "Hello"; /* s pointer */
```

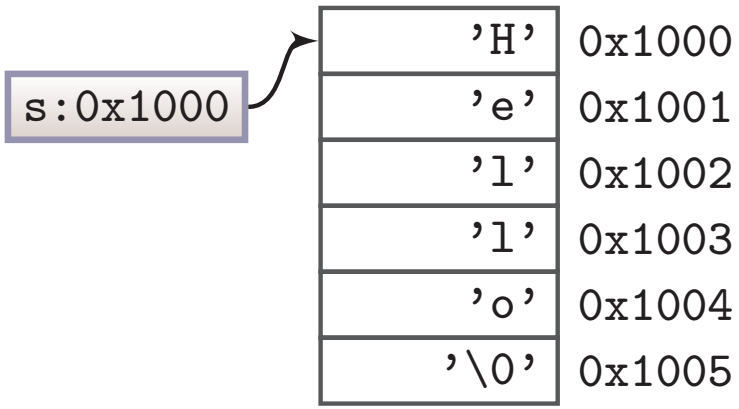


# Defining strings as character arrays



- Defining a constant character array and a pointer pointing to it, with initialization.

```
1  char *s = "Hello"; /* s pointer */
```



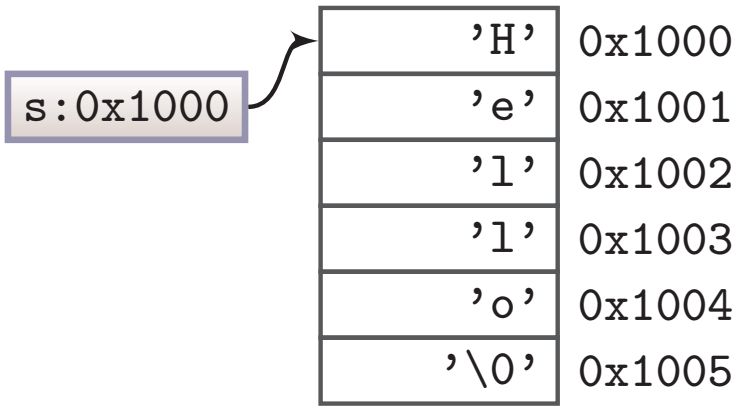

- Here the so-called static part of memory is used to store the string. The content of the string cannot be changed.

# Defining strings as character arrays



- Defining a constant character array and a pointer pointing to it, with initialization.

```
1  char *s = "Hello"; /* s pointer */
```




- Here the so-called static part of memory is used to store the string. The content of the string cannot be changed.
- We can modify value of s, however it is not recommended, because this stores the address of our string.

# Remarks

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Character or text?

```
1 char s[] = "A"; /* two bytes: {'A', '\0'} */
2 char c = 'A';   /* one byte: 'A' */
```

# Remarks



- Character or text?

```c
char s[] = "A"; /* two bytes: {'A', '\0'} */
char c = 'A';   /* one byte: 'A' */
```

- A text can be empty, but there is no empty character

```c
char s[] = "";  /* one byte: {'\0'} */
char c = '';    /* ERROR, this is not possible */
```

# Reading and displaying strings

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Strings are read and displayed with format code %s

```
1  char s[100] = "Hello";
2  printf("%s\n", s);
3  printf("Enter a word not longer than 99 characters: ");
4  scanf("%s", s);
5  printf("%s\n", s);
```

```
Hello
Enter a word not longer than 99 characters:  ghostbusters
ghostbusters
```

# Reading and displaying strings

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

- Strings are read and displayed with format code %s

```
1 char s[100] = "Hello";
2 printf("%s\n", s);
3 printf("Enter a word not longer than 99 characters: ");
4 scanf("%s", s);
5 printf("%s\n", s);
```

```
Hello
Enter a word not longer than 99 characters:  ghostbusters
ghostbusters
```

- Why don't we have to pass the size for `printf`?
- Why don't we need the `&` in the `scanf` function?

# Reading and displaying strings



- `scanf` reads only until the first whitespace character. To read text consisting of several words, use the `gets` function:

```
1 char s[100];
2 printf("Enter a text - max. 99 characters long: ");
3 gets(s);
4 printf("%s\n", s);
```

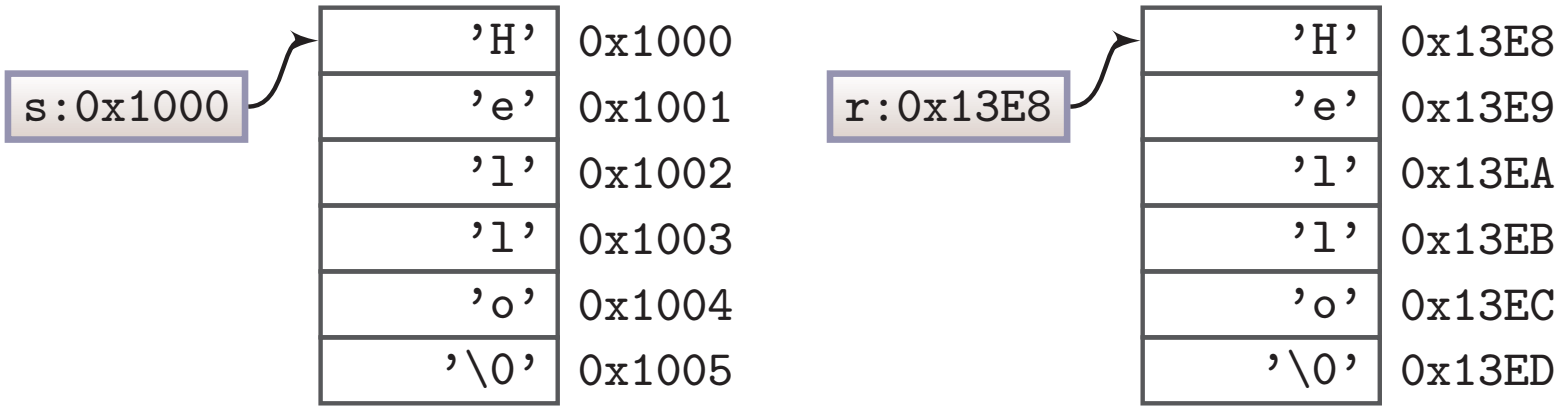
```
Enter a text - max.  99 characters long:  this is text
this is text
```

# Strings – typical mistakes



- Typical mistake: comparison of strings

```
1  char *s = "Hello";
2  char *r = "Hello";
3  if (s == r) /* what do we compare? */
4    ...
```

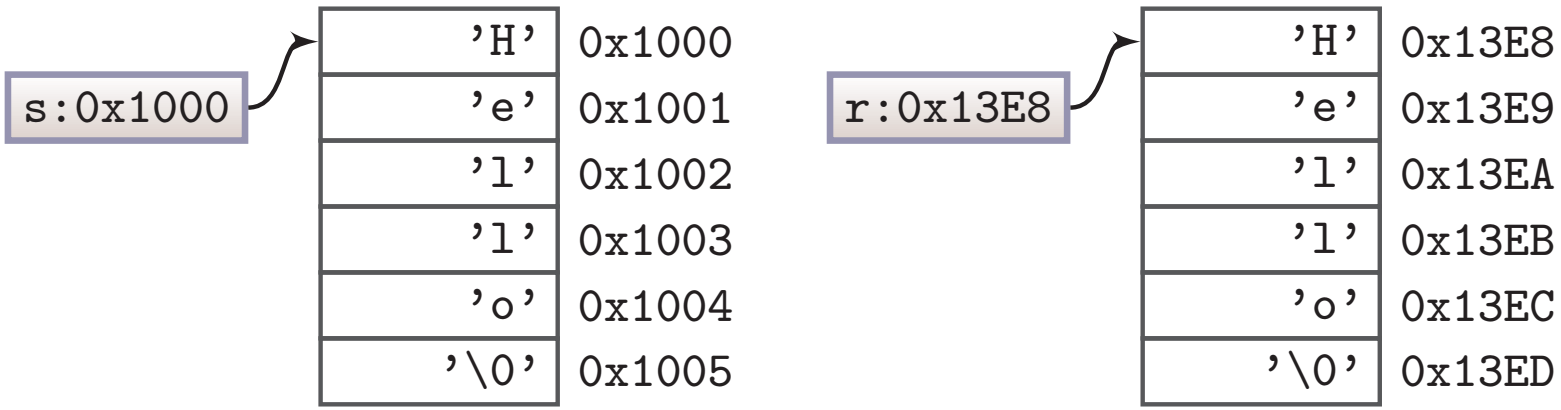


# Strings – typical mistakes



- Typical mistake: comparison of strings

```
1  char *s = "Hello";
2  char *r = "Hello";
3  if (s == r) /* what do we compare? */
4    ...
```




- The same mistake happens if defined as arrays

# String functions



- Comparing strings
- the result
  - positive, if s1 stands after s2 alphabetically
  - 0, if they are identical
  - negative, if s1 stands before s2 alphabetically

```
1  int strcmp(char *s1, char *s2) /* pointer-notation */
2  {
3    while (*s1 != '\0' && *s1 == *s2)
4    {
5      s1++;
6      s2++;
7    }
8    return *s1 - *s2;
9  }
```

# String functions



- Comparing strings
- the result
  - positive, if s1 stands after s2 alphabetically
  - 0, if they are identical
  - negative, if s1 stands before s2 alphabetically

```
1  int strcmp(char *s1, char *s2) /* pointer-notation */
2  {
3    while (*s1 != '\0' && *s1 == *s2)
4    {
5      s1++;
6      s2++;
7    }
8    return *s1 - *s2;
9  }
```

- Is it a problem, that s1 and s2 was changed during the check?

# String functions



- Comparing strings
- the result
  - positive, if s1 stands after s2 alphabetically
  - 0, if they are identical
  - negative, if s1 stands before s2 alphabetically

```
1  int strcmp(char *s1, char *s2) /* pointer-notation */
2  {
3    while (*s1 != '\0' && *s1 == *s2)
4    {
5      s1++;
6      s2++;
7    }
8    return *s1 - *s2;
9  }
```

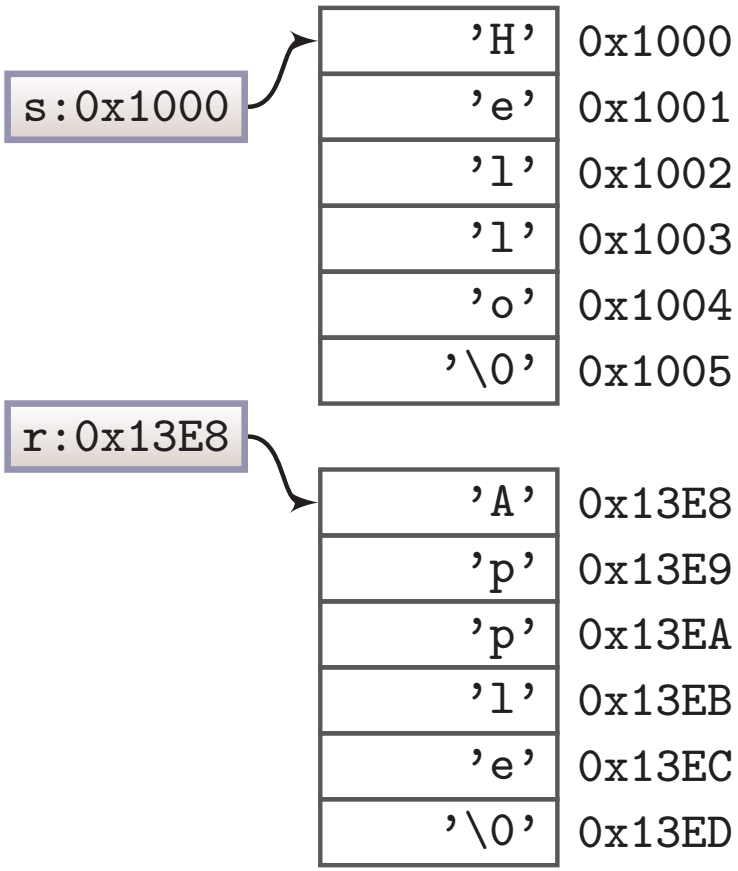
- Is it a problem, that s1 and s2 was changed during the check?
- Remark: In the solution we made use of the information that \0 is the 0 ASCII-code character!

# Strings – typical mistakes



- Typical mistake: string copy attempt

```
1 char *s = "Hello";
2 char *r = "Apple";
3 r = s; /* what do we copy */
```

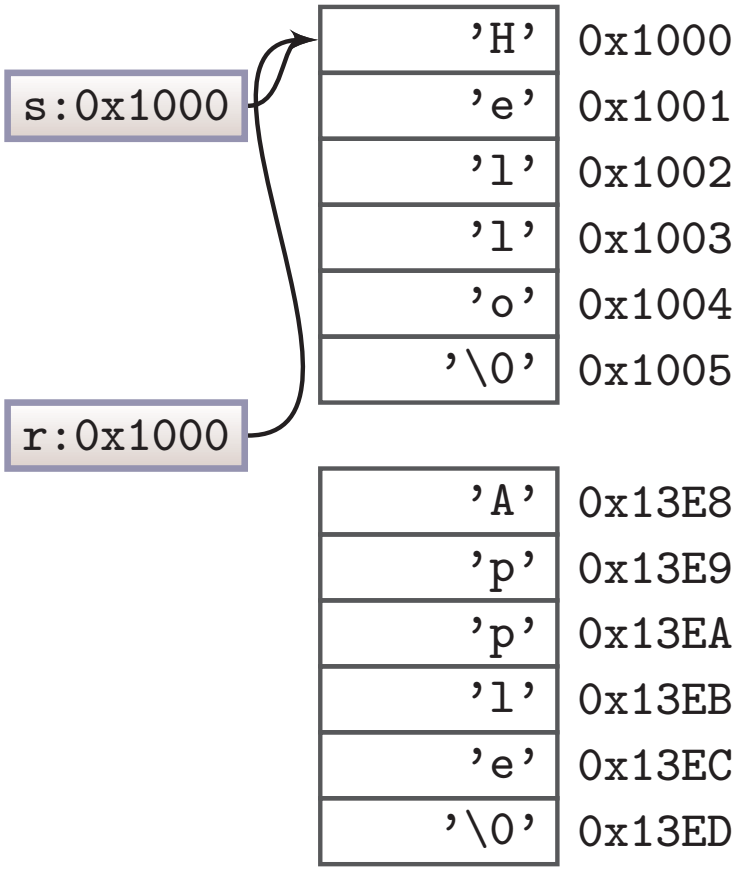


# Strings – typical mistakes



- Typical mistake: string copy attempt

```
1  char *s = "Hello";
2  char *r = "Apple";
3  r = s; /* what do we copy */
```



# Other string functions



- `#include <string.h>`
  - `strlen` length of string (without `\0`)
  - `strcmp` comparing strings
  - `strcpy` copying string
  - `strcat` concatenating strings
  - `strchr` search for character in string
  - `strstr` search for string in string
- `strcpy` and `strcat` functions copy 'without thinking', the user must provide the allocated memory for the resulting string!

Thank you for your attention.