Introduction to Mathematical Software 2008

TECHNISCHE UNIVERSITÄT DARMSTADT

Books:
The C++ Programming Language, by Bjarne Stroustrup
Programming in C, by Kerningham and Ritchie

Links:

http://www.cplusplus.com/
http://www.nongnu.org/c-prog-book/online/index.html
http://pweb.netcom.com/~tjensen/ptr/pointers.htm
http://www2.its.strath.ac.uk/courses/c/

Why C and C++?

TECHNISCHE UNIVERSITÄT DARMSTADT

- In the unix world, C is commonly used. Even the operating system itself is written in “C”.
- For many applications, C has a good ratio of understandability and efficiency; most remarkable software packages contain program parts which are written in C.
- Mathematical software often lives from its efficiency.
- Mathematical software is often dominated by algorithmic aspects. Collaborative and distributed development of the software, as well as aspects of large software systems are often minor important.
- Nevertheless: Object oriented programming as with C++ help us to structure our software.

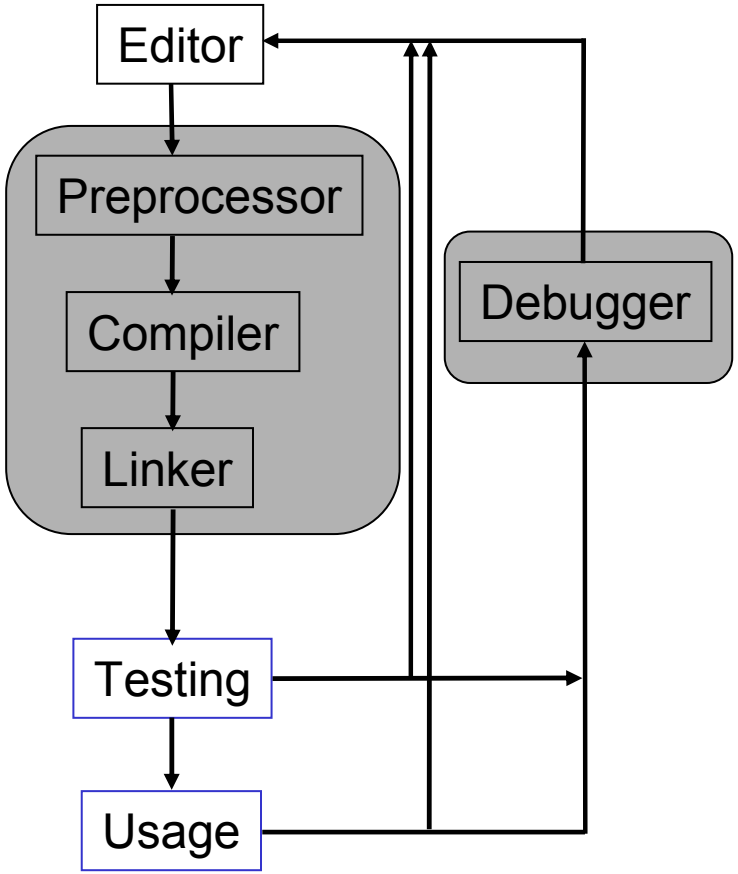
C/C++ programming, a very short overview

TECHNISCHE UNIVERSITÄT DARMSTADT

- Tools
  - editor, compiler and linker
- Programming language C
  - first examples
  - keywords
  - data: variables and their types
  - blocks, statements, and expressions
  - control structures: alternatives, loops, functions, recursion
- Object Orientation with C++
  - classes, objects, dynamic storage allocation
  - encapsulation and inheritance
- Algorithms: e.g. sorting

Tools

TECHNISCHE UNIVERSITÄT DARMSTADT



Editor
- source files,
  Tool: e.g. xemacs: www.xemacs.org

Preprocessor, Compiler, Linker
- translate the source code of a high level language, given in textfile(s), to machine language, and binds (links) them together
  Tool: e.g. gcc, g++

Debugger
- allows to observe the program while it is running.
  Tool: e.g. gdb

turnaround cycles in program development

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example:

```
#include <stdio.h>
int main(void)
{
  int year = 2008;
  printf("Hello World %d!\n",year);
  return 0;
}
```

compilation: g++ hello.cc –o hello
start: ./hello
output: Hello World!

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Keywords:

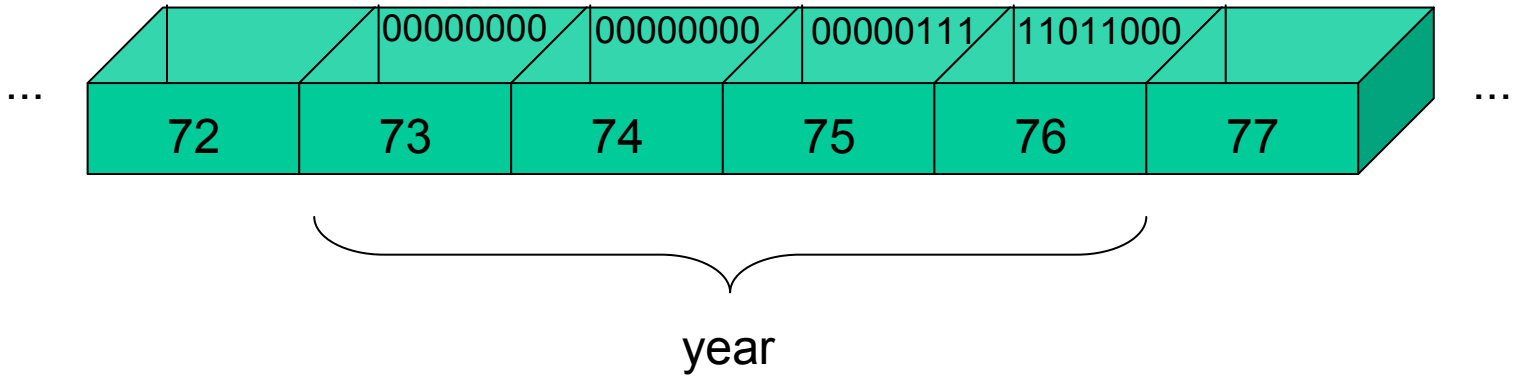
auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Variables:

Variables reserve memory in the main memory. Each variable gets a name.



Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Variables, visibility:

- global variables are accessible everywhere; are defined outside any block
- local variables have a scope. They are only valid in those blocks, where they have been defined.
  Examples:

```
{
 int i;
 i = 2;
 {
   int j,k;
   j = 3;
   k = i + j;
 }
 i = k; // Error!
}
```

```
{
 int i, j;
 i = 0;
 {
   int i;
   i = 1; //new definition of i
          //shadows old one
 }
 j = i; // j = 0
}
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

The type of a variable determines how much memory is reserved.

	sign	smallest value	largest value	size in bits
char	yes	-128	127	8
unsigned char	no	0	255	8
short	yes	-32768	32767	16
unsigned short	no	0	65535	16
long	yes	-2147483648	2147483647	32
unsigned long	no	0	4294967295	32
float	yes	$1.2 * 10^{-28}$	$3.4 * 10^{38}$	32
double	yes	$2.32 * 10^{-308}$	$1.8 * 10^{308}$	64
int	yes	depends on machine and OS		
unsigned int	no			
size_t	no			
ssize_t	yes			
off_t	no			
long long	yes	-2^{63}	$2^{63} -1$	64
unsigned long long	no	0	$2^{64} - 1$	64
long double	yes	$3.4 * 10^{-4932}$	$1.2 * 10^{4932}$	96

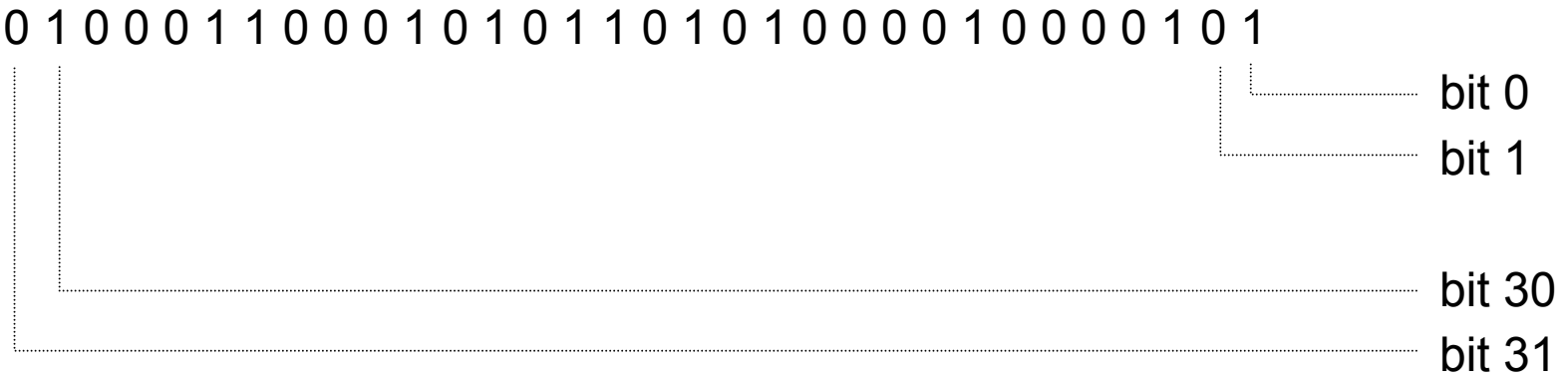
Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Interpretation of the types

integer variables are the most natural and mostly used kind of variables

Bitstrings are interpreted as numbers in the dual number system.



The value then is $bit31 \cdot 2^{31} + bit30 \cdot 2^{30} \ldots bit0 \cdot 2^{0}$.
Negative numbers are^realized with the help of the twos-complement.

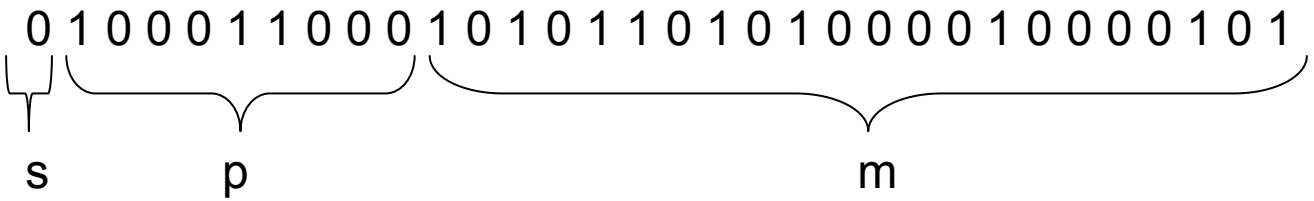
Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Interpretation of the types

floating point variables

0/1 sequences are interpreted as sign (s) , mantissa (m) and exponent (p)



the resulting number then is $s \cdot m \cdot 2^p$

This is not exactly the way, how the bit masks are interpreted, but the principle is similar. For details see e.g. http://www.ieee.org.

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Interpretation of the types

character variables are used for letters

Bitstrings are firstly interpreted as 8-bit integer number and then looked up in the so-called ASCII table, in order to interpret that number as a letter.

0 1 0 0 0 0 0 1 is 65, and char(65) is ‘A’.

For further details, see e.g. http://www.asciitable.com

TECHNISCHE UNIVERSITÄT DARMSTADT

Pointers and arrays

The statement

int a[10];

defines an array with 10 elements:

a : 

a[0] a[1] ... a[9].

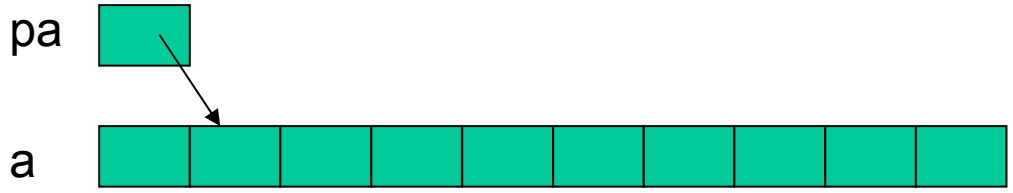
a[i] is the i-th element of the array, counting from 0.

int *pa; defines a ‘pointer’ to an integer variable.

We can initialize it e.g. with

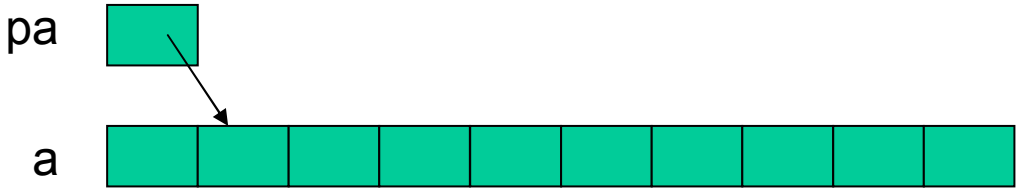
pa = &a[1];

This means, pa holds the memory address of a[1].

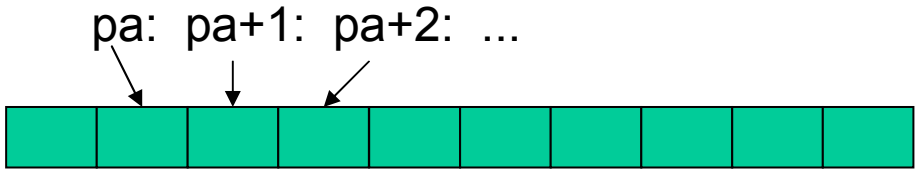


Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT



x = *pa; dereferences pa and assigns a[1] to x.



This works independently of the array's / pointer's data type:

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int ai[10]; long double ald[10]; char ac[10];
int *pi;    long double *pld;    char *pc;

pi = &ai[3]; pld = &ald[3]; pc = &ac[3];
printf("pi=%p, pld=%p, pc=%p\n",pi,pld,pc);

pi = &ai[4]; pld = &ald[4]; pc = &ac[4];
printf("pi=%p, pld=%p, pc=%p\n",pi,pld,pc);

printf("#bytes: int:%d, long double:%d, char:%d\n",
        (unsigned int)(pi+1)-(unsigned int)pi,
        (unsigned int)(ald+1)-(unsigned int)ald,
        (int)(pc+1)-(int)pc);
```

```
bash-3.2$ ./small_pointer_arithmetics.exe
pi=0x22ccbc, pld=0x22cc54, pc=0x22cc23
pi=0x22ccc0, pld=0x22cc60, pc=0x22cc24
#bytes: int:4, long double:12, char:1
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Strings in C are nothing else than zero-terminated arrays of char.

"Hello World"-String in memory:

H	e	l	l	o		W	o	r	l	d	'\0'
72	101	108	108	111	32	87	111	114	108	100	0

```
int strlen(char *s)
{
      int n;
      for (n = 0; *s != '\0';s++)
          // same as s != 0, not the same as s != '0'
          n++;
      return n;
}

ac[0] = '0';
printf(" %c %d %d\n",ac[0],ac[0],strlen("Hello World");
```

generates the following output:

0 48 11

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Memory allocation

static global variable allocation: outside of any procedure

- global: accessible from everywhere
- static: is fixed with the start of the program

local variable allocation: inside blocks or functions

- only accessible in the currently alive blocks

dynamic allocation: with the help of malloc(size_t s) and free()

e.g.

```
int *a = (int*)malloc(sizeof(int) * 1000);
//use a in the same way as an array
free(a);
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Memory allocation

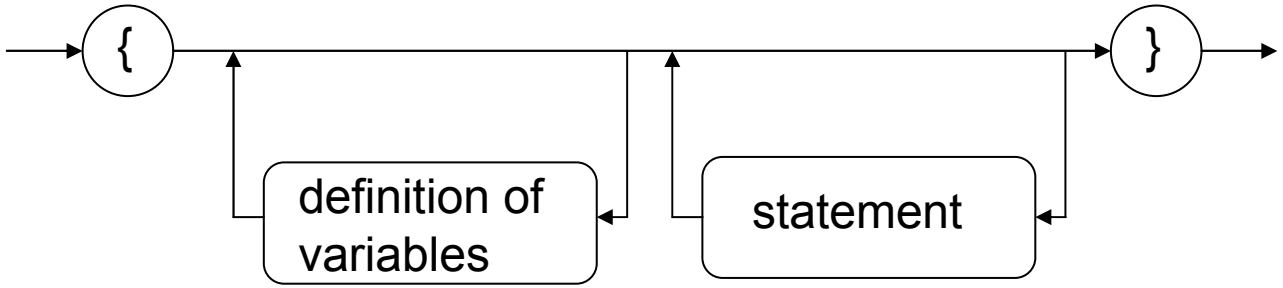
On a stack we only have access to the latest inserted data.

malloc returns a memory block, which is
- logically consecutive
- may physically have holes (not important for us)

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Blocks: a block is a sequence of statements, grouped by brackets

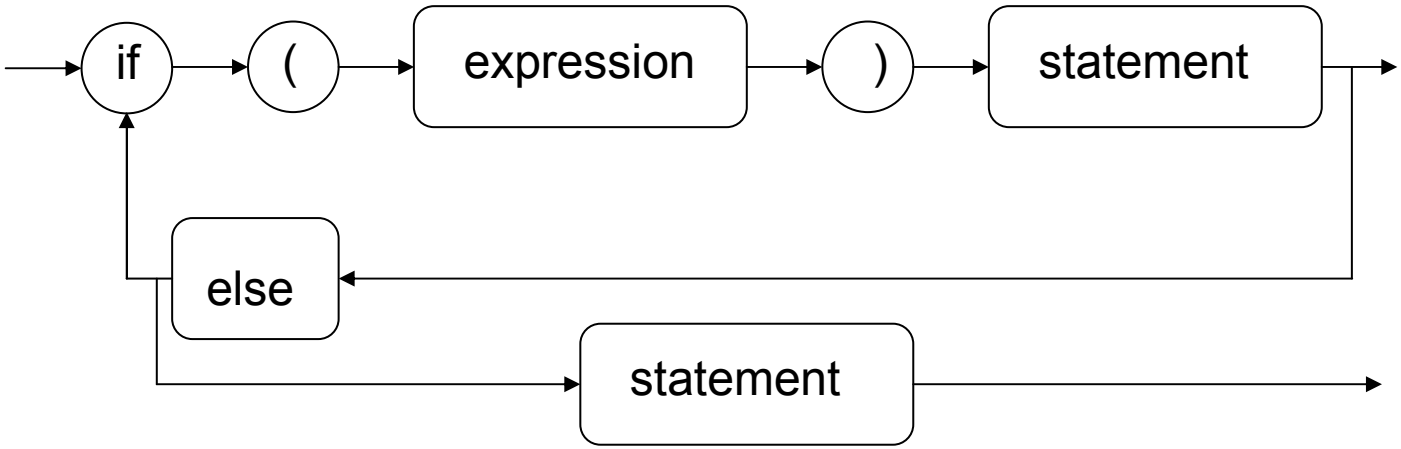


Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Flow control:

Alternative program flow



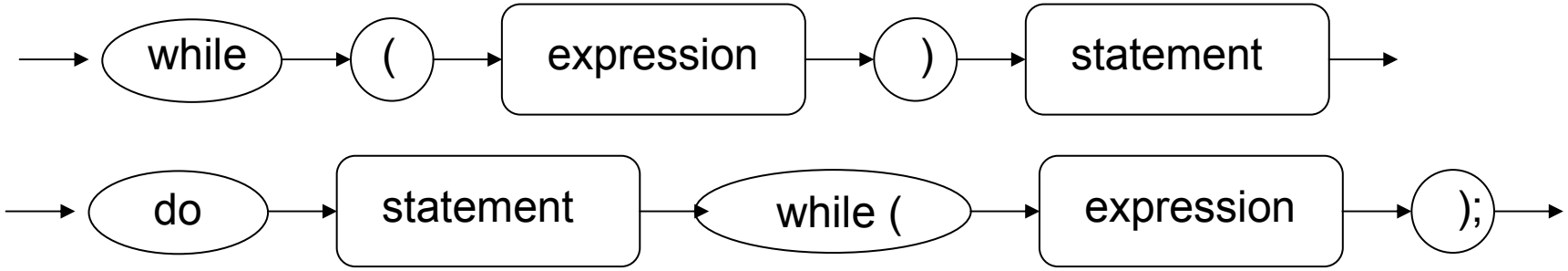

```
if (i < 3) printf("i is smaller than 3");
else if (i > 3) printf("i is larger than 3.");
else printf("i is equal to 3.");
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Flow control:

while loops




```
while (i < 3) {
   printf("i is %d",i);
   i = i + 1;
   i++;
}
```

```
do {
   printf("i is %d",i);
   i = i + 1;
   i++;
} while (i < 3);
```

Remark:

```
expr1; while (expr2) { statemant; expr3; }
```

is equivalent to

```
for (expr1; expr2; expr3) statement;
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

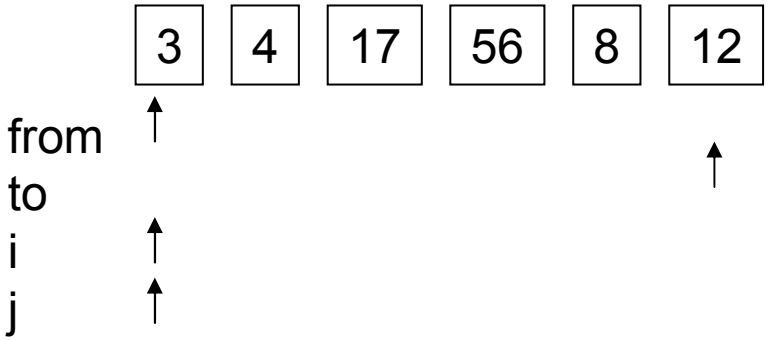
Let ‘a’ be an array of integer variables, ‘from’ and ‘to’ be two indices into that

```
{
  int i,j;
  for (i = 0; i <= to;i++)
     for(j = i;j <= to;j++)
        if (a[i] > a[j]) {
           int buffer;
           buffer = a[i];  a[i] = a[j];  a[j] = buffer;
        }
}
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

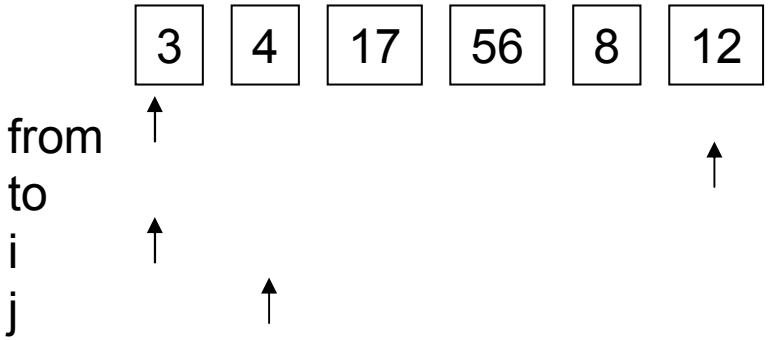



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

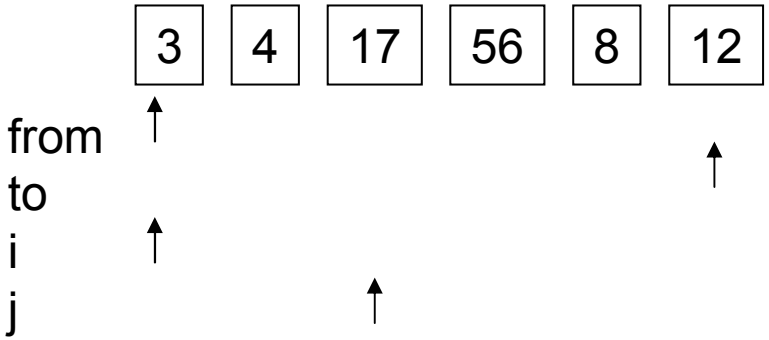



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

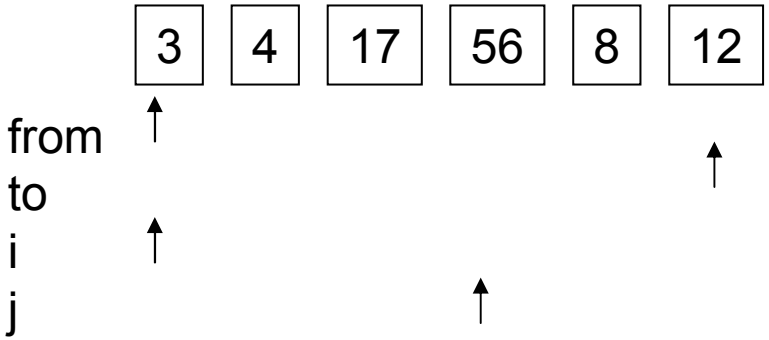



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

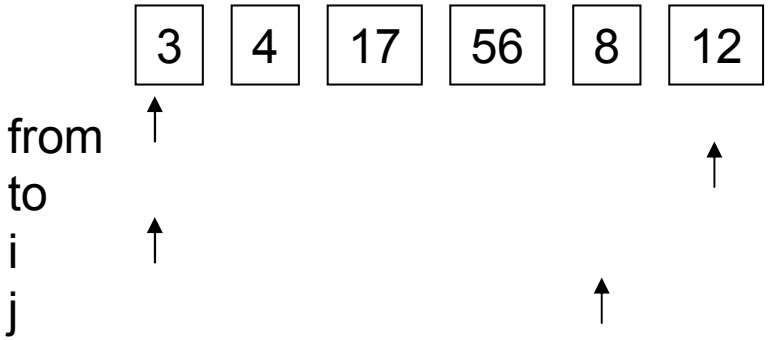



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

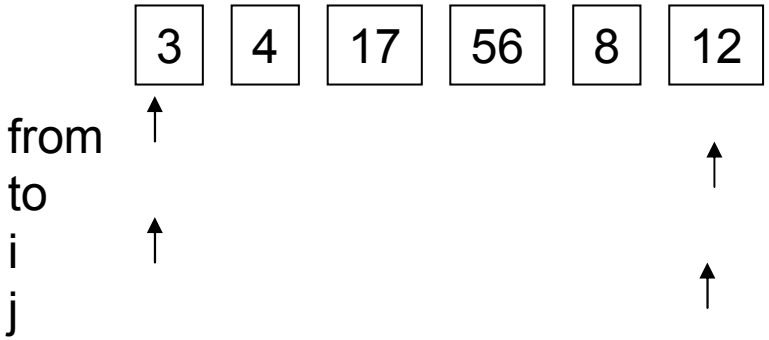



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

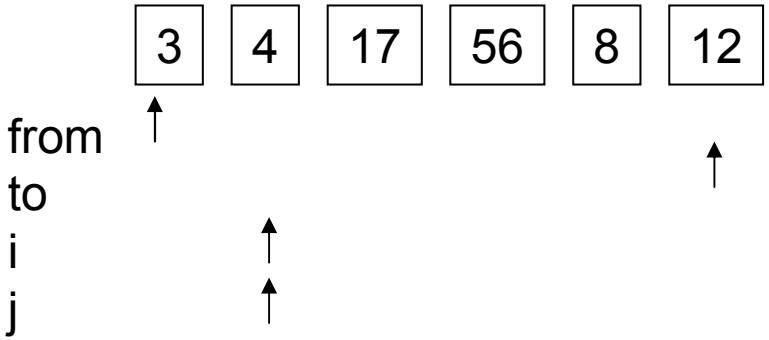



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

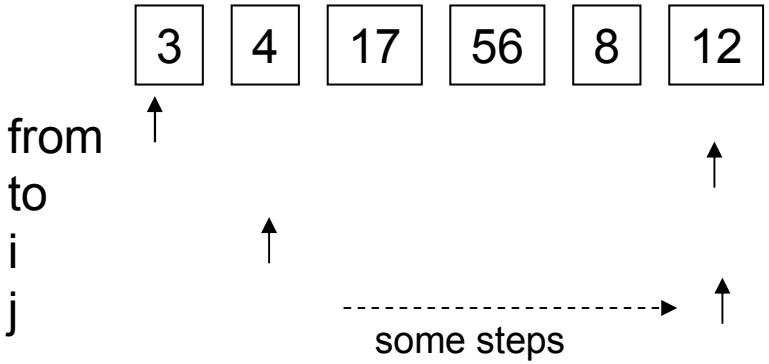



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

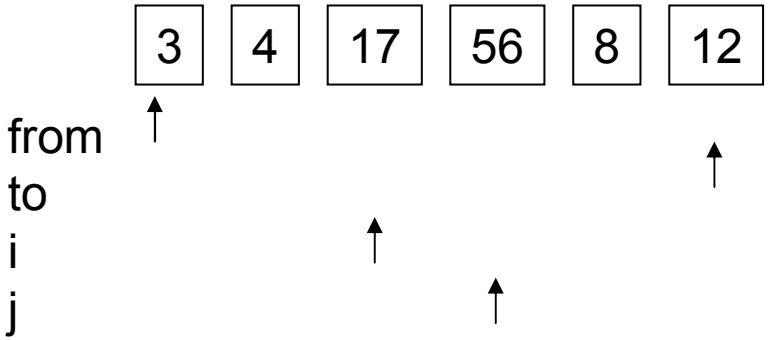



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

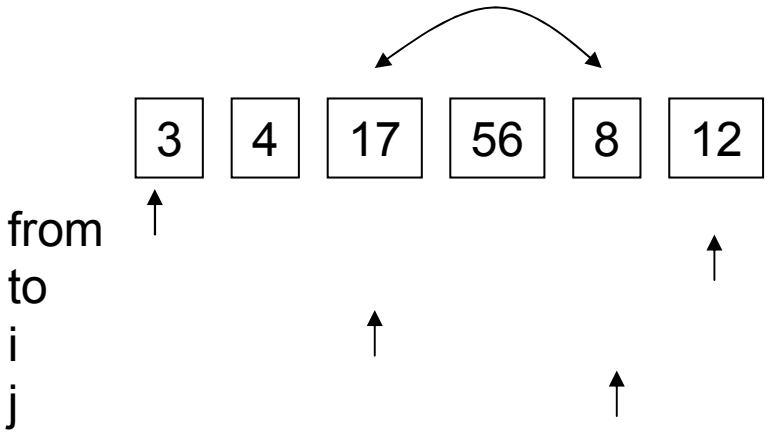



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

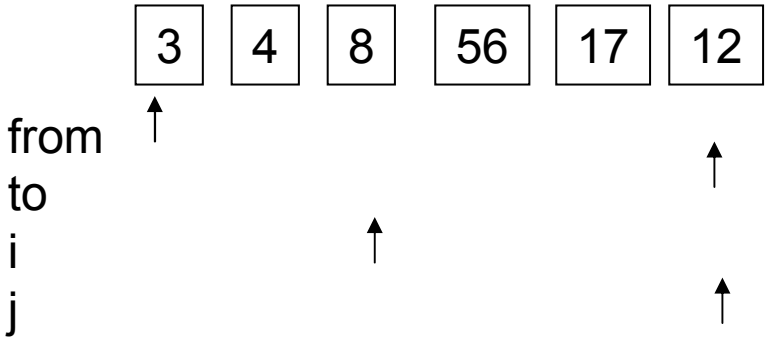



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

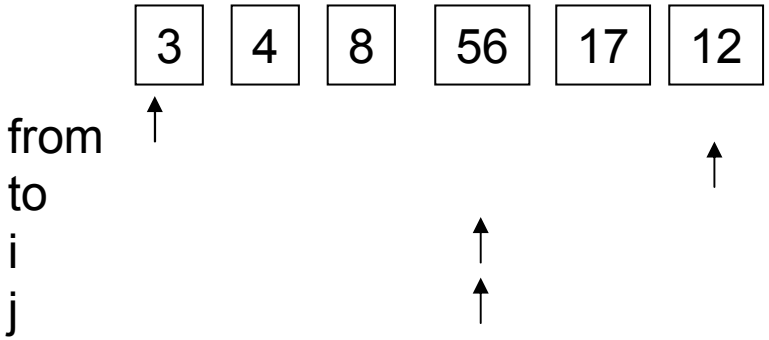



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

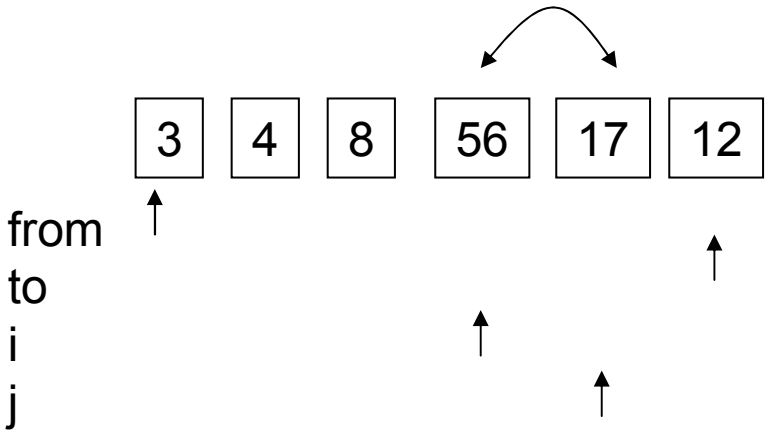



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

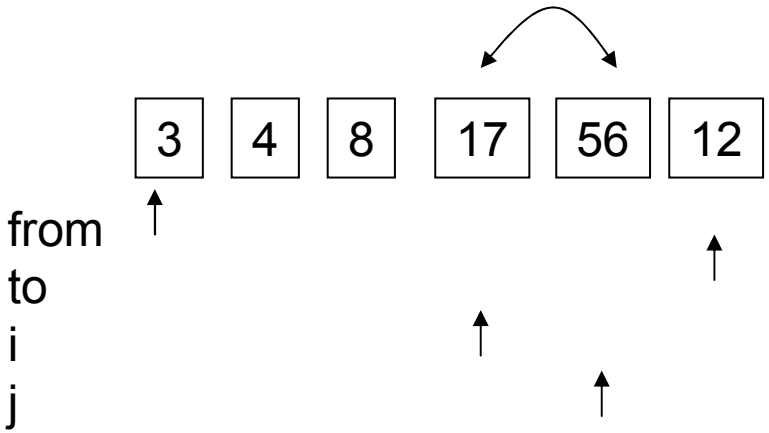



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

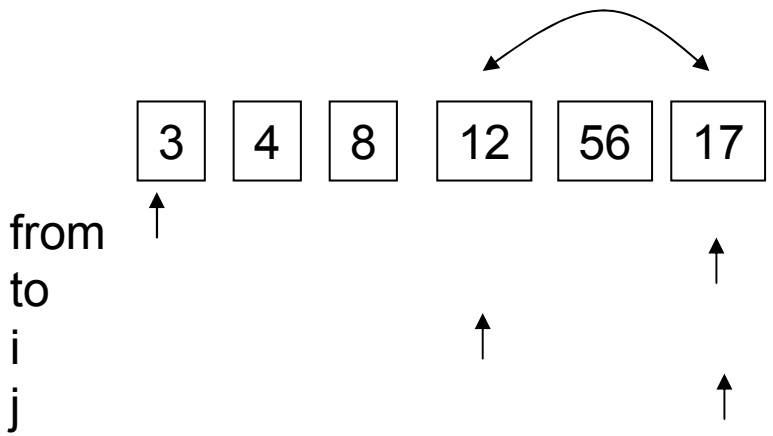



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

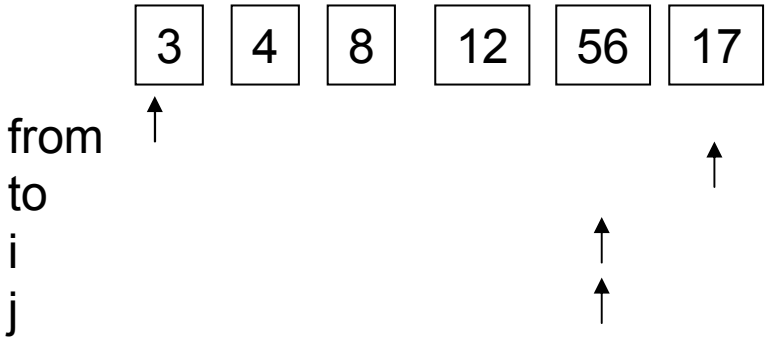



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

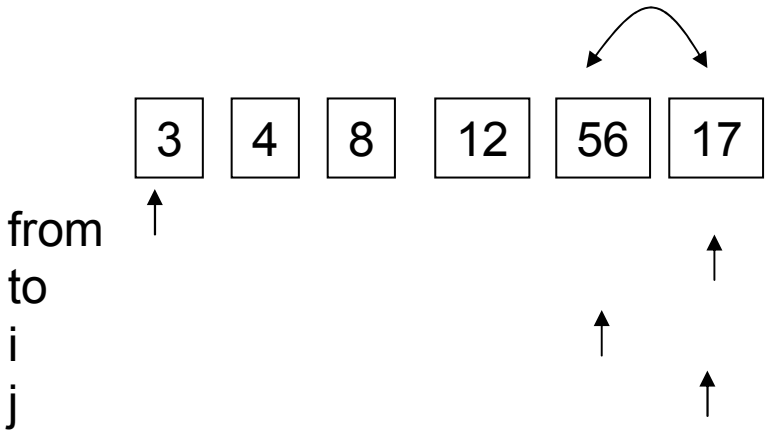



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

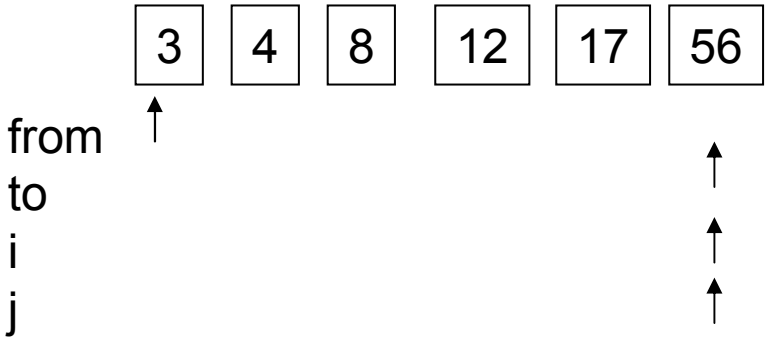



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

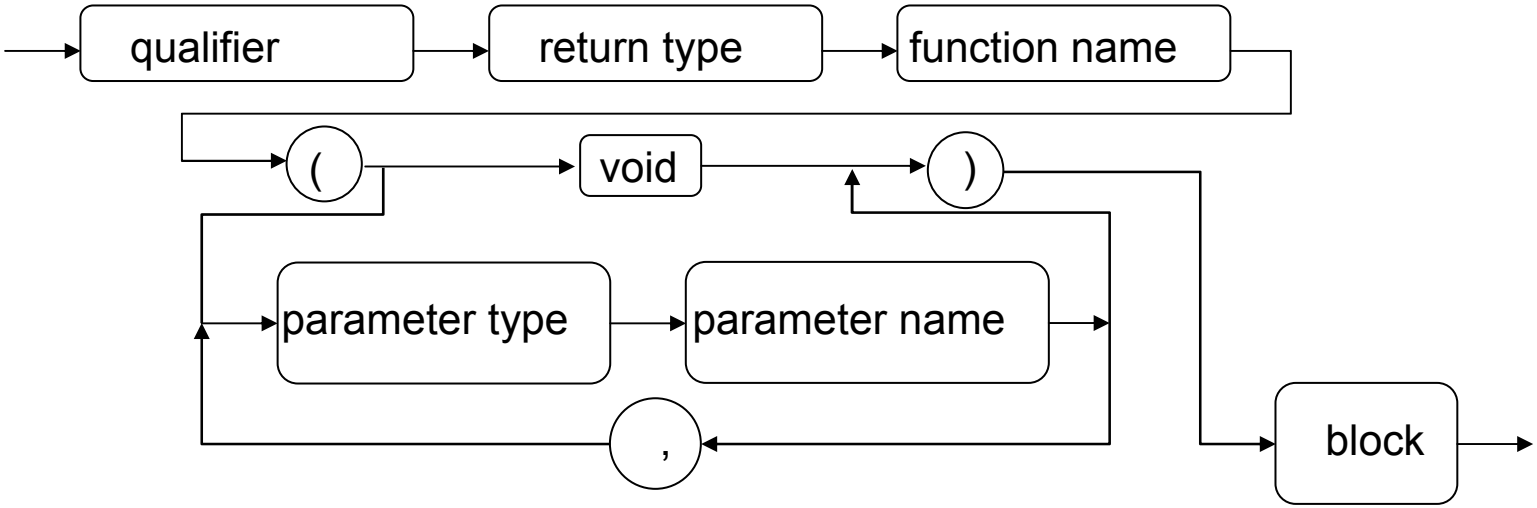



```
int i,j;
for (i = 0; i <= to;i++)
   for(j = i;j <= to;j++)
      if (a[i] > a[j]) {
         int buffer;
         buffer = a[i];  a[i] = a[j];  a[j] = buffer;
      }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Functions (procedures, methods), --> named blocks




```
int square(int x)
{
    int result = x*x;
    return result;
}
```

```
usage in main:
....
  int a;
  a = square(x);
...
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Bubblesort:

Input: Array ‘a’ of integer variables, a first index which indicates the first entry to be sorted in the array, and a last index, which indicates the last entry in the array.

```
void  bubblesort(int *a,int from,int to)
{
  int i,j;
  for (i = 0; i <= to;i++)
     for(j = i;j <= to;j++)
        if (a[i] > a[j]) {
           int buffer;
           buffer = a[i];  a[i] = a[j];  a[j] = buffer;
        }
}
```

Analysis:
-outer loop i from 0 to ‘to‘.
- inner loop from i to ‘to‘
- no more than

$$\sum_{i=0}^{n}\sum_{j=i}^{n} 3 = \frac{3}{2}n^2 + \frac{9}{2}n + 3$$

steps.
- $O(n^2)$ steps

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Mergesort:

Input: Array ‘a’ of integer variables, a first index which indicates the first entry to be sorted in the array, and a last index, which indicates the last entry in the array.

```
void  mergesort(int *a,int l,int r)
{
  int i,j,k,m, b[r+1];

  if (r > l) {
     m = (r+l)/2;
     mergesort(a,l,m);
     mergesort(a,m+1,r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m  ;j < r; j++) b[r+m-j] = a[j+1];
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
  }
}
```

Analysis:
-T(1) = 3
-T(n) = 2*T(n/2) + 2*log(n)
-rsolve({ T(1) = 3, T(n) = 2 T(n/2) + 2*n }, T)
= n (3 ln(2) + 2 ln(n)) / ln(2)
- O(n log(n))

Programming language C

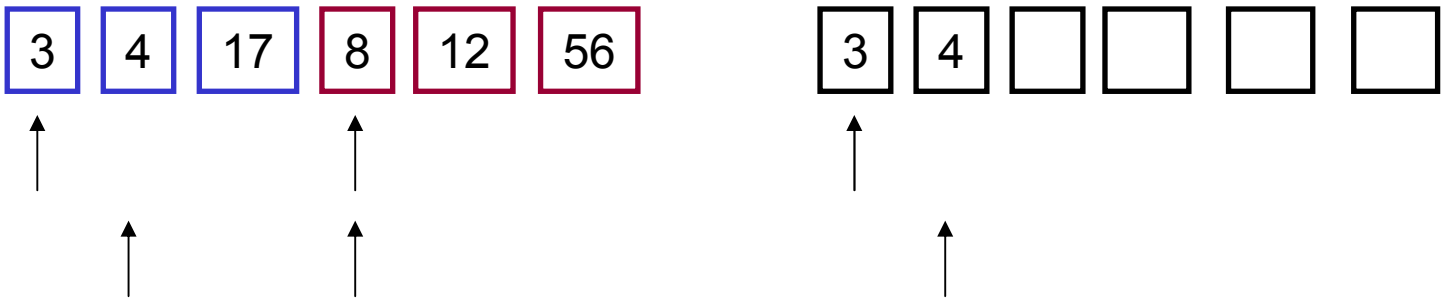
TECHNISCHE UNIVERSITÄT DARMSTADT

Example, merge-operation

Let two sorted arrays be given. The merge-operation combines them to one sorted array in linear time.

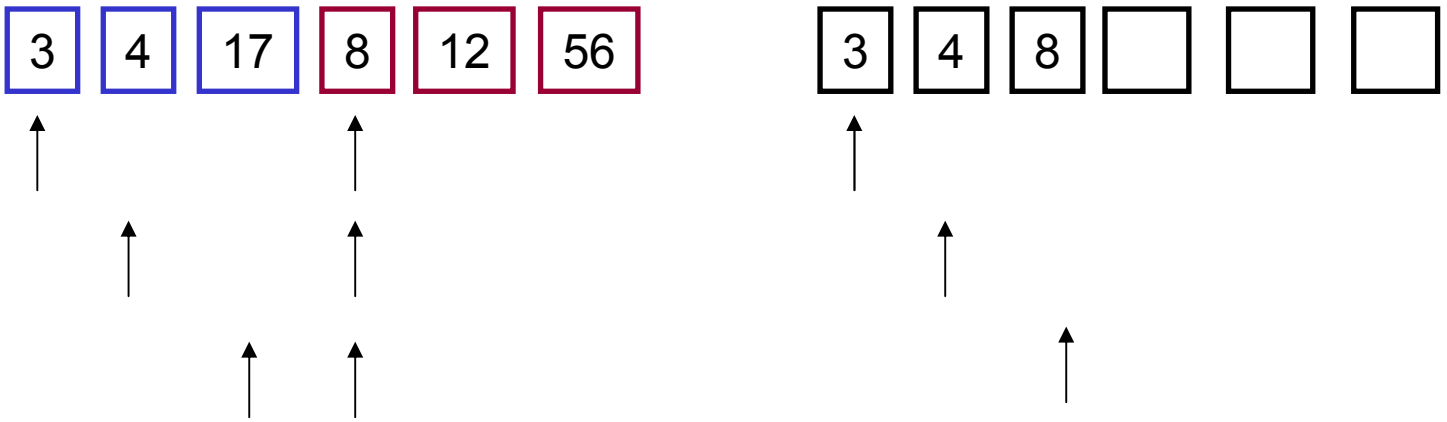
Example, merge-operation

Let two sorted arrays be given. The merge-operation combines them to one sorted array in linear time.



Example, merge-operation

Let two sorted arrays be given. The merge-operation combines them to one sorted array in linear time.

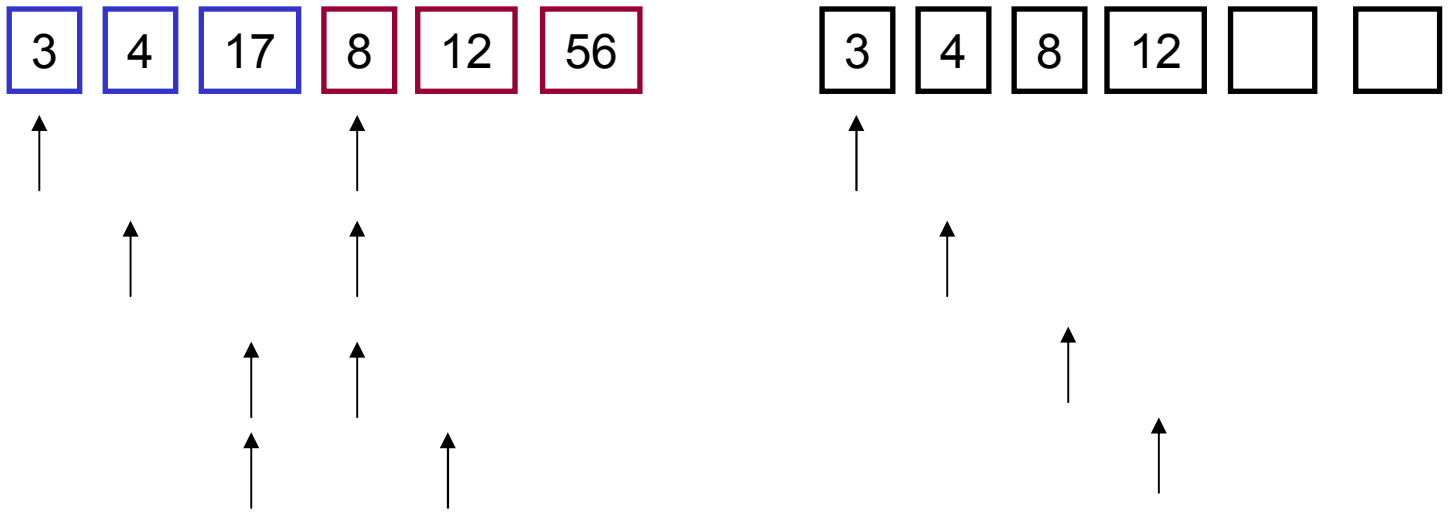


Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example, merge-operation

Let two sorted arrays be given. The merge-operation combines them to one sorted array in linear time.

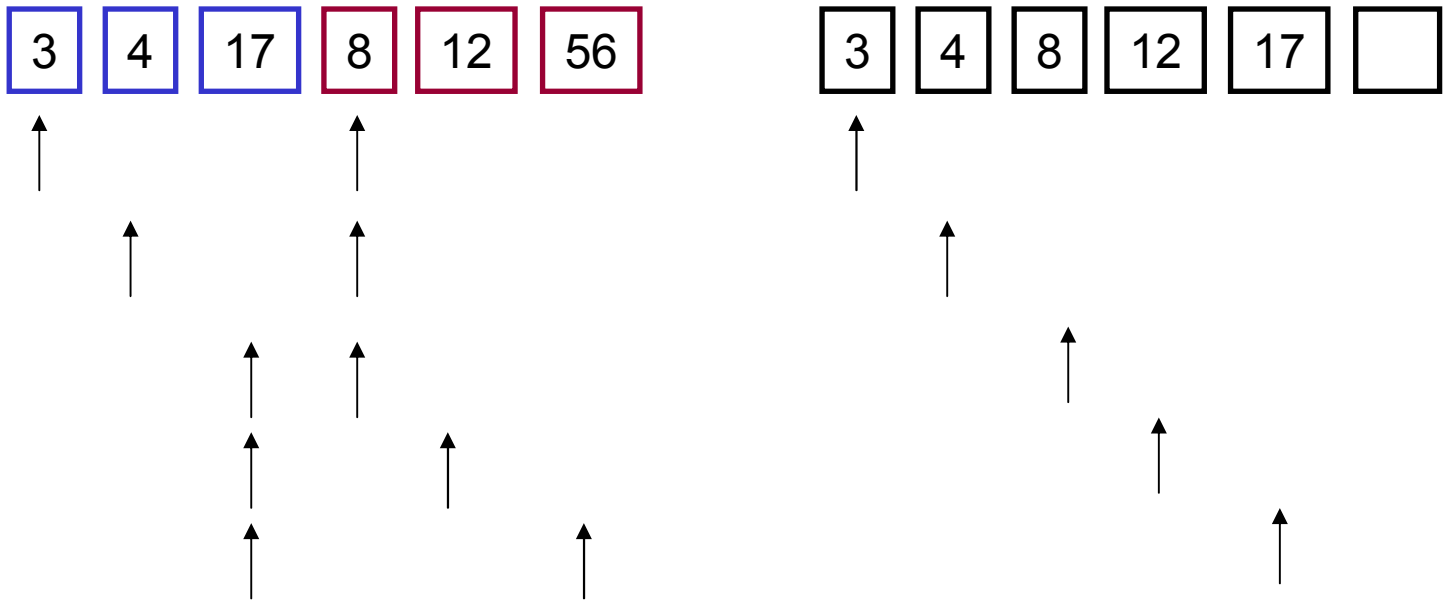


Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example, merge-operation

Let two sorted arrays be given. The merge-operation combines them to one sorted array in linear time.

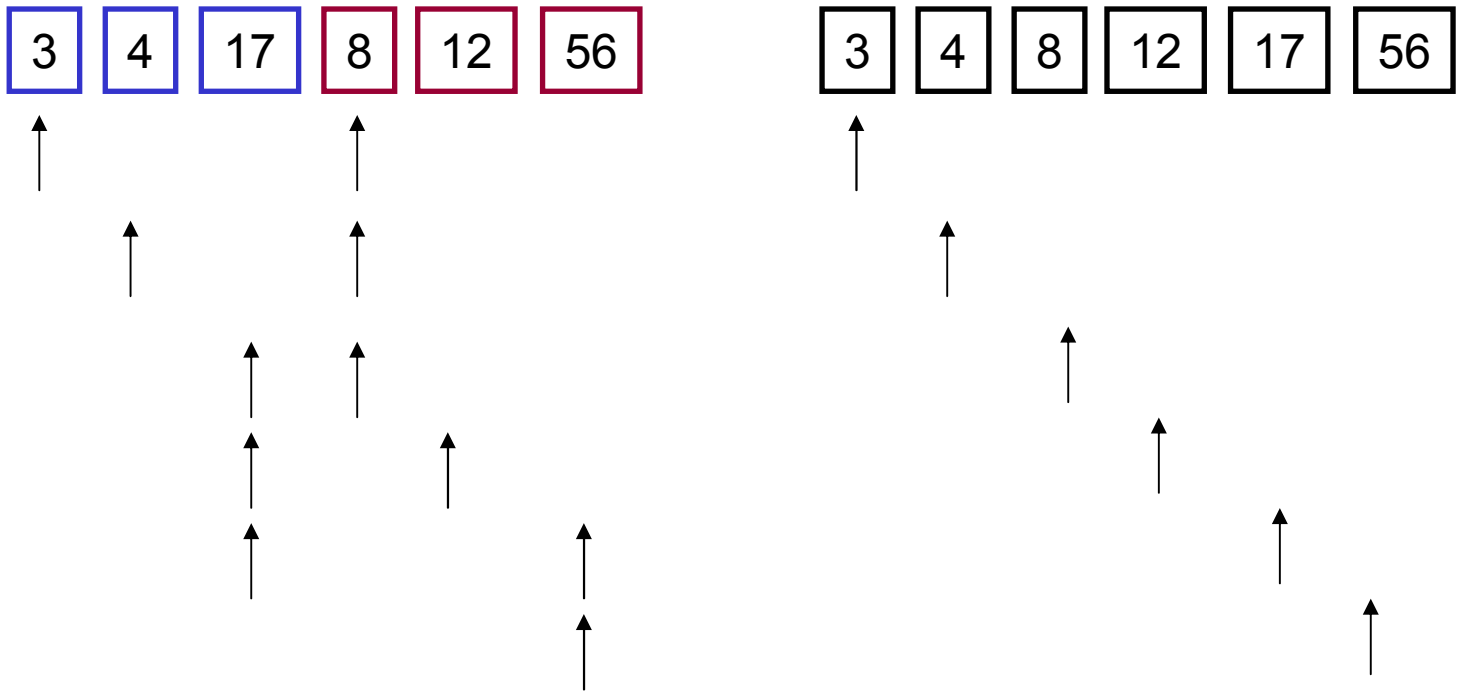


Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example, merge-operation

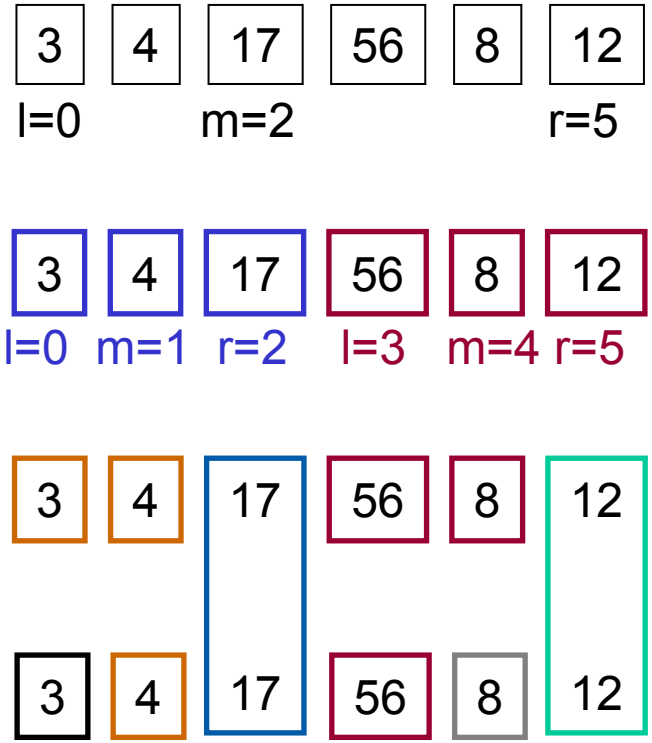
Let two sorted arrays be given. The merge-operation combines them to one sorted array in linear time.



Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

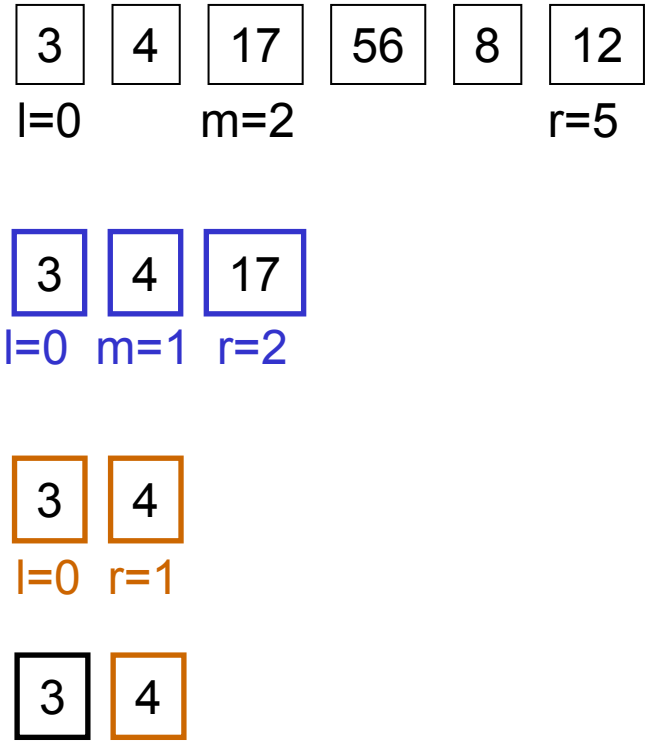



```
void  mergesort(int *a,int l,int r)
{
  int i,j,k,m, b[r+1];
  if (r > l) {
     m = (r+l)/2;
     mergesort(a,l,m);
     mergesort(a,m+1,r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m  ;j < r; j++) b[r+m-j] =
a[j+1];
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
  }
 }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

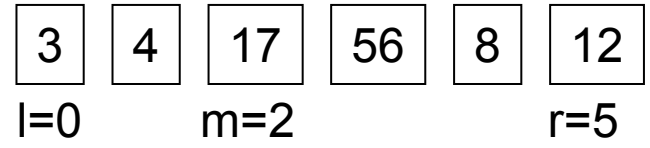


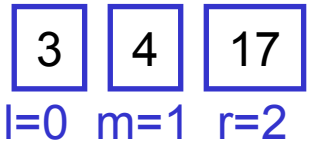

```
void  mergesort(int *a,int l,int r)
{
  int i,j,k,m, b[r+1];
  if (r > l) {
     m = (r+l)/2;
     mergesort(a,l,m);
     mergesort(a,m+1,r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m  ;j < r; j++) b[r+m-j] =
a[j+1];
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
  }
 }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example



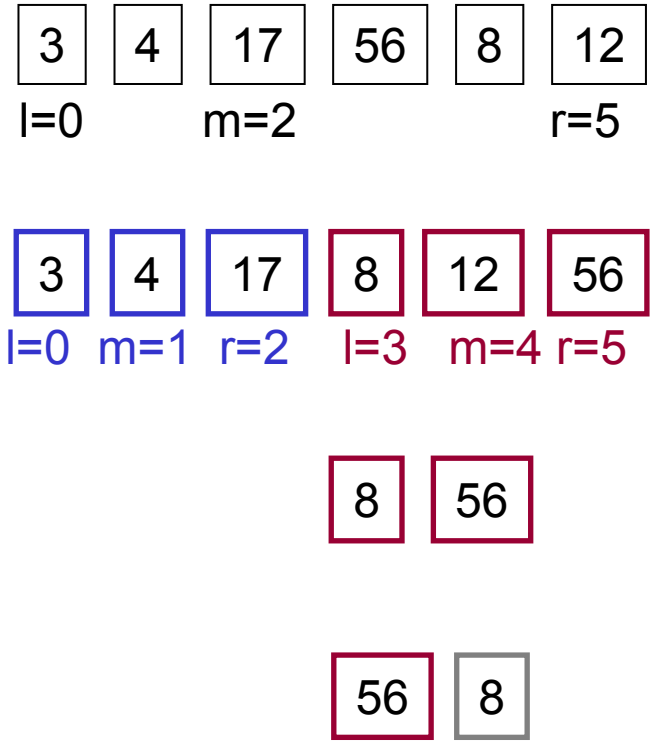



```
void  mergesort(int *a,int l,int r)
{
  int i,j,k,m, b[r+1];
  if (r > l) {
     m = (r+l)/2;
     mergesort(a,l,m);
     mergesort(a,m+1,r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m  ;j < r; j++) b[r+m-j] =
a[j+1];
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
  }
 }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example

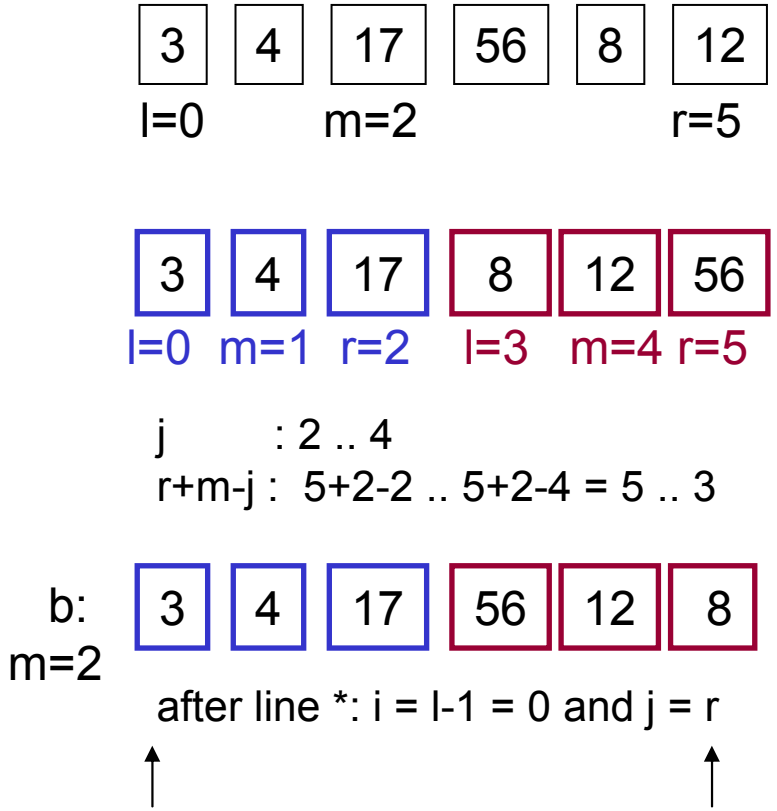



```
void  mergesort(int *a,int l,int r)
{
  int i,j,k,m, b[r+1];
  if (r > l) {
     m = (r+l)/2;
     mergesort(a,l,m);
     mergesort(a,m+1,r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m  ;j < r; j++) b[r+m-j] =
a[j+1];
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
  }
 }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Example




```
void  mergesort(int *a,int l,int r)
{
  int i,j,k,m, b[r+1];
  if (r > l) {
     m = (r+l)/2;
     mergesort(a,l,m);
     mergesort(a,m+1,r);
     for (i = m+1; i > l; i--) b[i-1] = a[i-1];
     for (j = m  ;j < r; j++) b[r+m-j] = a[j+1]; //
line *
     for (k = l; k <= r; k++)
       a[k] = (b[i] < b[j]) ? b[i++] : b[j--];
  }
 }
```

Programming language C

TECHNISCHE UNIVERSITÄT DARMSTADT

Bucketsort: Assumption: Let the maximum number that can occur be known as MAX_VAL. Let n be the length of the array, which we want to sort.

Input: Array ‘a’ of integer variables, a first index which indicates the first entry to be sorted in the array, and a last index, which indicates the last entry in the array.

```
void  bucketsort(int *a,int l,int r)
{
  int buffer[MAX_VAL],i,j;
  for (i = 0; i < MAXVAL; i++) buffer[i] = 0;
  for (i = l; i < r; i++) {
     buffer[a[i]]++;
  }
  for (i = 0, j = l; i < MAXVAL; i++)
     if (buffer[i] != 0)
        for (;buffer[i]--;j++) {
             a[j] = i;
        }
}
```

Analysis:

- first loop from 0 to MAXVAL.
- second loop from l to r
- third loop from 0 to MAXVAL
- O(MAXVAL + n) steps

A little bit object orientation

TECHNISCHE UNIVERSITÄT DARMSTADT

classes

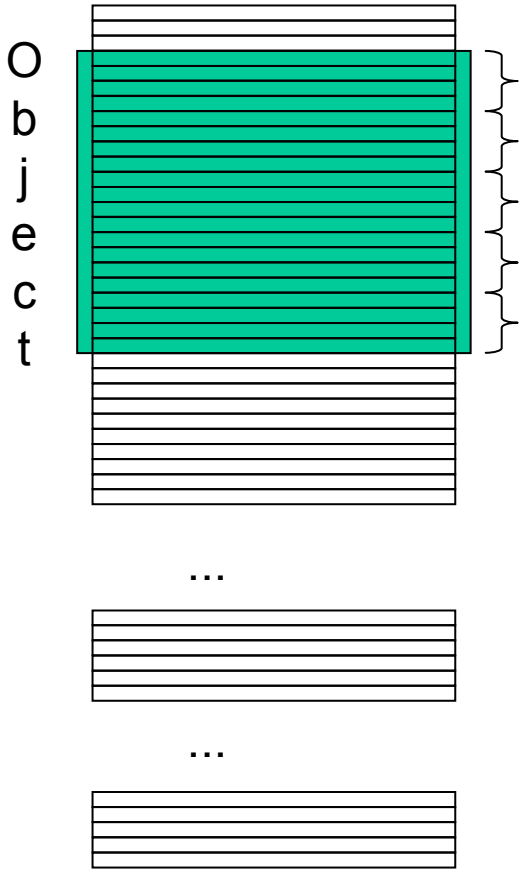
A class binds several variables and functions together.

```
class Fraction {
    int enumerator;
    int denominator;
    Fraction(int e, int d) { enumterator = e; denominator = d;}
    ~Fraction() {}
    Fraction *product(Fraction *a, Fraction *b)
     {
          return new Fraction(a->enumerator*b->enumerator,
                                   a->denominator*b->denominator);
     }
};
```

A shade of object orientation

TECHNISCHE UNIVERSITÄT DARMSTADT

objects are instantiated classes, i.e. a class is the type of a object variable



int enumerator;
int denominator;
pointer to Fraction *product(Fraction *a, Fraction *b)
pointer to Fraction(int e, int d)
pointer to ~Fraction()

```
Fraction *product(Fraction *a, Fraction *b)
 {
   return new Fraction(a->enumerator*b->enumerator,
                       a->denominator*b->denominator);
 }
```

A shade of object orientation

TECHNISCHE UNIVERSITÄT DARMSTADT

Inheritance

A parent class inherits its properties (variables and functions) to a child class.

```
class betterFraction : Fraction {
   betterFraction *sum(betterFraction *a, betterFraction *b)
    {
        betterFraction *nf = new betterFraction(
                                        a->enumerator*b->denominator +
                                        b->enumerator*a->denominator,
                                        a->denominator*b->denominator);
        return nf;
    }
};
```