

Vistula, IT Faculty, 2014

# Algorithms and Complexity

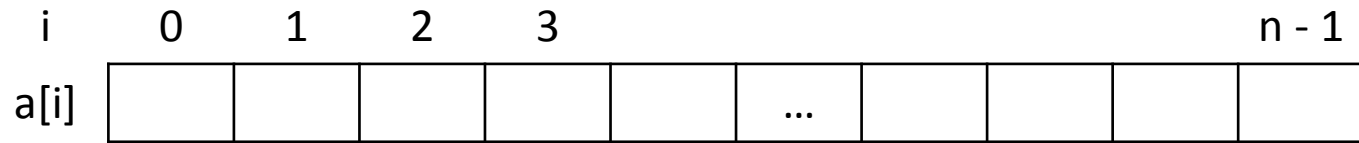
Dmitry A. Zaitsev

<http://daze.ho.ua>

## Lecture 6:

# Arrays, tables and lists

# One dimension array



Mapping to computer memory:

$\text{address}[i] = \text{base\_address} + i * \text{sizeof}(\text{element\_type})$

Static array:

```
#define N 10
int a[N];
int i;

for(i=0; i<N; i++)
    a[i]=5;
```

Dynamic array:

```
int n,i;
int *a;
scanf("%d", &n);
a=malloc( n * sizeof(int) );
for(i=0; i<N; i++)
    a[i]=5;
free(a);
```

References to elements:

```
a[i]
*(a +i)
```

# Matrix

$i \setminus j$	0	1	...	m-1
0			...	
1				
...	...	...	...	...
n-1			...	

Mapping to computer memory:

$\text{address}[i][j] = \text{base\_address} + (i * m + j) * \text{sizeof}(\text{element\_type})$

Static matrix:

```
#define N 10
#define M 15
int a[N][M];
int i, j;

for(i=0; i<N; i++)
    for(j=0; j<M; j++)
        a[i][j]=5;
```

Dynamic matrix:

```
int n,m,i,j;
int *a;
scanf("%d %d", &n, &m);
a=malloc( MATRIX_SIZE(n,m,int) );
for(i=0; i<n; i++)
    for(j=0; j<m; j++)
        MELT(a,i,j,n,m)=5;
free(a);
```

## Matrix access macroses

```
#define MATRIX_SIZE(d1,d2,t) ((d1)*(d2)*(sizeof(t)))
```

```
#define MOFF(i,j,d1,d2) ((d2)*(i)+(j))
```

```
#define MELT(x,i,j,d1,d2) (*((x)+MOFF(i,j,d1,d2)))
```

Examples of substitution

```
MATRIX_SIZE(10,20,int) ((10)*(20)*(sizeof(int)))
```

```
MOFF(3,4,10,20) ((20)*(3)+(4))
```

```
MELT(a,3,4,10,20) (*((a)+((20)*(3)+(4))))
```

## d-dimension arrays

```
#define D1 10
```

```
#define D2 15
```

```
...
```

```
#define Dd 25
```

```
int a[D1][D2]...[Dk];
```

```
int i1, i2,...,id;
```

```
for(i1=0; i1<D1; i1++)
```

```
    for(i2=0; i2<D2; i2++)
```

```
        ...
```

```
            for(id=0; id<D2; id++)
```

```
                a[i1][i2]...[id]=5;
```

## Tables (structures)

	surname	name	weight	hight
0				
1				
...				
n-1				

```
struct person {  
    char surname[N+1];  
    char name[N+1];  
    int weight;  
    int height;  
};
```

```
struct person * sp;  
scanf("%d",&n);  
sp=malloc(n*sizeof(struct person));  
for(i=0;i<n;i++)  
{  
    scanf("%s %s %d %d",sp[i].surname,sp[i].name,&(sp[i].weight),&(sp[i].height));  
}  
for(i=0;i<n;i++)  
{  
    printf("%-16s %-16s %3d %3d\n",sp[i].surname,sp[i].name,sp[i].weight,sp[i].height);  
}
```

## Search in tables

A key – a field to provide access (search)

Sequential search ( $O(n)$ )

```
found=0;
for(i=0;i<n;i++)
{
    if(strcmp(argv[1],sp[i].surname)==0){found=1;i1=i;break;}
}

if(found)printf("\n\n%-16s %-16s %3d %3d\n",
                sp[i1].surname,sp[i1].name,sp[i1].weight,sp[i1].height);
```

# Sparse arrays (matrices)

An array of structures  
for nonzero elements:

```
struct sparse_matrix {  
    int i;  
    int j;  
    int aij;  
};
```

Lines or columns with nonzero elements –  
require variable length arrays

k	i	j	aij
0			
...			
L-1			



## Sets and bags

$$A = \{a, b, c\}, B = \{a, b, d, e\}$$

$$C = A \cup B = \{a, b, c, d, e\}$$

U – a universal set

$$U = \{a, b, c, d, e, f, g, h\}$$

Set is represented with a bit array

	<b>a</b>	<b>b</b>	<b>c</b>	<b>d</b>	<b>e</b>	<b>f</b>	<b>g</b>	<b>h</b>
bB	1	1	0	1	1	0	0	0

# Abstract lists

$l1 = [1, 5, 4, 8, 9]$

$l2 = [a, b, u, d, k, d, g, a]$

Basic operations:

$[e]$  – constructor of a list

$hd\ l$  – head of a list

$tl\ l$  – tail of a list

$l1 \wedge l2$  – concatenation of two lists

$nil$  - an empty list

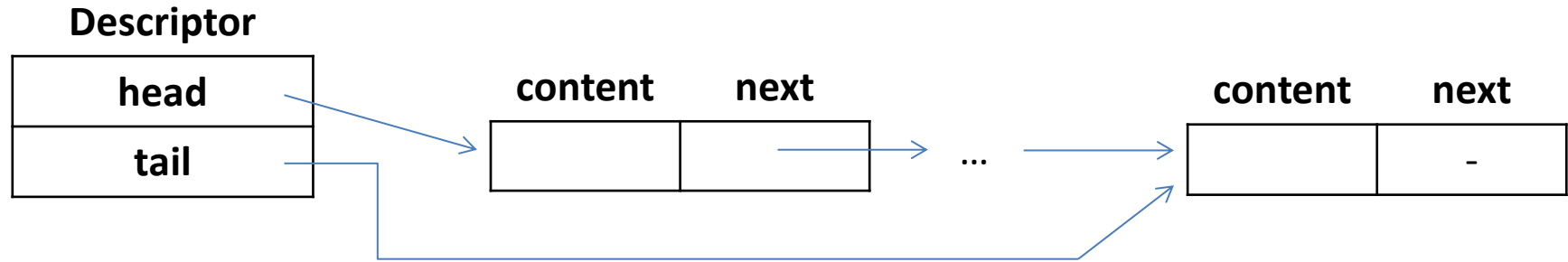
$hd\ l1 = 1$

$tl\ l1 = [1, 5, 4, 8, 9]$

$l1 \wedge [3, 6] = [1, 5, 4, 8, 9, 3, 6]$

Languages: LISP, ML

# Linked lists (one link)



```
struct elist1 {  
    int cont;  
    struct elist1 * next;  
};
```

```
struct dlist1 {  
    struct elist1 * head;  
    struct elist1 * tail;  
}
```

# FIFO (queue)

```
in_fifo(struct dlist1 * q, int c)
{
    struct elist1 * e = malloc(sizeof(struct elist1));
    e->cont=c;
    e->next=NULL;
    if( q->head == NULL )
    {
        q->head = e;
        q->tail = e;
    }
    else
    {
        (q->tail)->next = e;
        q->tail=e;
    }
}
```

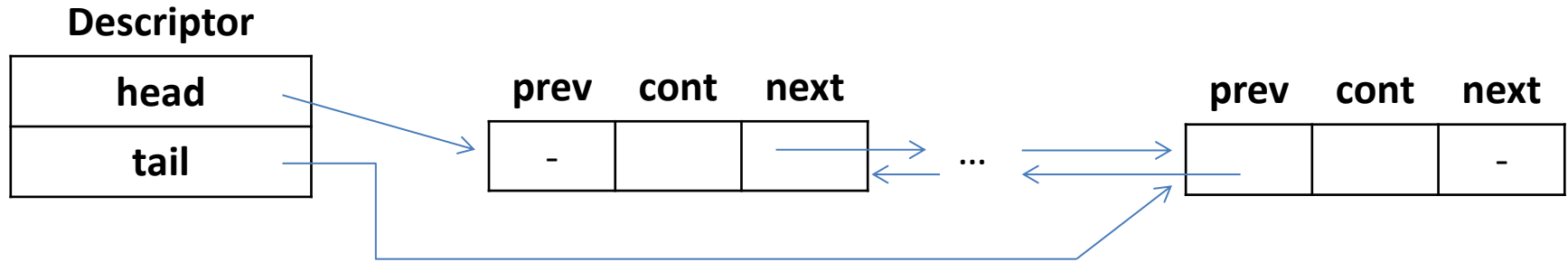
```
int out_fifo(struct dlist1 * q)
{
    struct elist1 * e;
    int c;
    if( q->head == NULL )
    {
        e = NULL;
    }
    else
    {
        e = q->head;
        q->head = e->next;
        e->next = NULL;
        if(q->tail==e) q->tail=NULL;
    }
    if(e==NULL) c=0; else { c=e->cont; free(e); }
    return(c);
}
```

# LIFO (stack)

```
in_fifo(struct dlist1 * q, int c)
{
    struct elist1 * e = malloc(sizeof(struct elist1));
    e->cont=c;
    e->next=NULL;
    if( q->head == NULL )
    {
        q->head = e;
        q->tail = e;
    }
    else
    {
        (q->tail)->next = e;
        q->tail=e;
    }
}
```

```
int out_fifo(struct dlist1 * q)
{
    struct elist1 * e;
    int c;
    if( q->head == NULL )
    {
        e = NULL;
    }
    else
    {
        e = q->head;
        q->head = e->next;
        e->next = NULL;
        if(q->tail==e) q->tail=NULL;
    }
    if(e==NULL) c=0; else { c=e->cont; free(e); }
    return(c);
}
```

# Two link lists



```
struct elist2 {  
    struct elist2 * prev;  
    int cont;  
    struct elist2 * next;  
};
```

```
struct dlist2 {  
    struct elist2 * head;  
    struct elist2 * tail;  
}
```