Tehnici de programare avansate

2022

ovidiu.banias@upt.ro

Tehnici de programare

Analiza algoritmilor

Pointeri. Lucru pe biți

Stiva

Backtracking

Recursivitate

Divide at Impera

Greedy

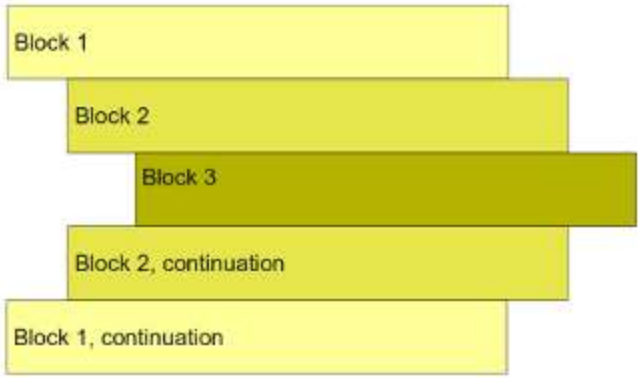
Programare dinamică

Bibliografie selectivă

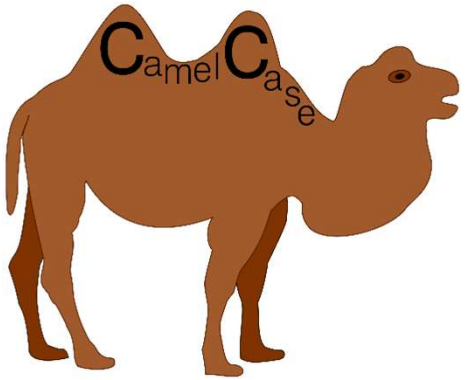
- Tudor Sorin, *Tehnici de programare*, Teora, 1995
- Cormen T.H., Leiserson C.E., Rivest R.R., *Introducere în algoritmi*, Agora, 2000
- Ivaşc C., Prună M., *Bazele informaticii*, Petrion, 1995
- Atanasiu. A, Pintea R., *Culegere de probleme pascal*, Petrion, 1996
- Manz. D, et. al., *Informatica. Culegere de probleme rezolvate şi propuse*, Mirton, 2005
- Ionescu C., et. al., *Informatica pentru grupele de performanță*, Dacia Educațional, 2004
- Ciocârlie H., Ciocârlie R., *Tehnici de programare şi structuri de date*, Eurostampa, 2010
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-00sc-introduction-to-computer-science-and-programming-spring-2011/
- http://videolectures.net/mit6046jf05_introduction_algorithms/
- https://class.coursera.org/algo-004/lecture

Principii generale.Identare/Notații cod/Comentare

Identare (K&R, KNF, GNU)



Notații cod (Camel Case / Hungarian)



Comentare

Analiza algoritmilor

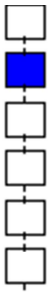
Problemă

Se dă o listă de n elemente, $A[1..n]$ și un element auxiliar x.

Să se găsească indexul i, cu proprietatea că $A[i]=x$, $1 \leq i \leq n$.

- **exemple**
- **2 metode**

Căutare liniară



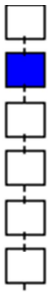
Algoritm: LINEARSEARCH

Input: vector A[1..n] de n elemente și un element x.

Output: i dacă x = A[i], 1 ≤ i ≤ n, și 0 în caz contrar.

- câte comparații minim?
- câte comparații maxim?

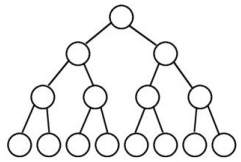
Căutare liniară



Algoritm: LINEARSEARCH

```
1. i ← 1

2. while (i < n) and (x ≠ A[i])

3.     i ← i+1

4. end while

5. if (x = A[i]) then return i else return 0
```

Căutare binară



Algoritm: BINARYSEARCH

Input: vector A[1..n] de n elemente sortate crescător și elementul x

Output: i dacă x == A[i], 1 ≤ i ≤ n, și 0 în caz contrar.

Observație: fie **li** și **lf** doi indecși ai vectorului **A[1..n]**, reprezentând limita inițială și respectiv limita superioară a unui subvector **A[li..lf]**.

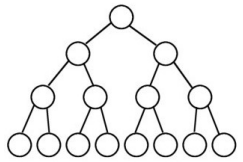
Se observă că:

A[li] ≤ A[(li+lf)/2] ≤ A[lf]

Rezolvare: la fiecare pas se va împărți intervalul în jumătate și elementul **x** se va căuta în doar unul din cele două intervale de indecși **A[li..(li+lf)/2]** şi **A[(li+lf)/2+1..lf]**, în funcție de valoarea **A[(li+lf)/2]** .

- câte comparații minim? câte comparații maxim?

Căutare binară



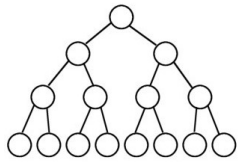
Algoritm: BINARYSEARCH – exemplificare (element căutat x=37)

Index (i)	1	2	3	4	5	6	7	8	9	10
A[1..10]	3	7	12	21	25	26	27	29	37	39

- li=1, lf=10
- pas 1: k=[(li+lf)/2] = 5
- pas 2: se obțin 2 intervale [1..5] și [6..10]
- pas 3: pt. că A[k]=25 < 37 se alege intervalul A[6..10]
- pas 4: li=6, lf=10, repeta pasul 1 până (li==lf) sau A[k]==x

Căutare binară

Algoritm: BINARYSEARCH – exemplificare (element căutat x=37)



Index (i)	6	7	8	9	10
A[6..10]	26	27	29	37	39

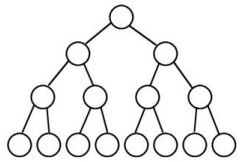
k=8, (A[k]=29)<37→ A[8..10]

Index (i)	8	9	10
A[8..10]	29	37	39

k=9, A[k]=37→ OK

http://www.infoarena.ro/blog/cautare-binara

Căutare binară

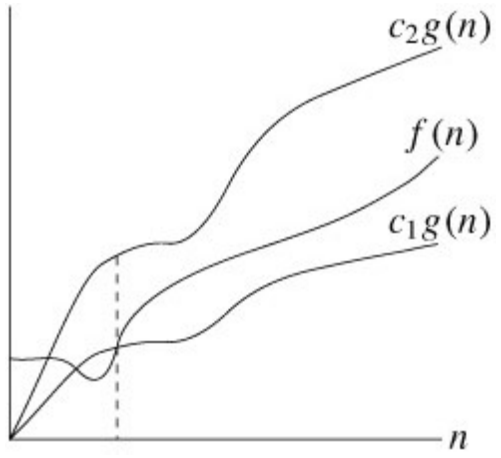


Algoritm: BINARYSEARCH – pseudocod

```
1. li ← 1; lf ← n;
2. while (li ≤ lf)
3.    k ← [(li + lf) / 2]
4.    if (x = A[k]) then return 1
5.    else if (x < A[k]) then lf ← k-1
6.                       else li ← k+1
7. end while
8. return 0
```

- după fiecare pas i, numărul de elemente se înjumătățeşte: pas i → $\frac{n}{2^{i-1}}$ elemente
- la ultimul pas, $\left\lfloor \frac{n}{2^{i-1}} = 1 \right\rfloor \Rightarrow i = \lfloor \log_2 n + 1 \rfloor$

Notații asimptotice



O limita sperioară

Θ limita inferioară == limita superioară

Ω limita inferioară

Notații asimptotice - O

Definiție:

fie $f(n), g(n) : \mathrm{N} \to R$

spunem că complexitatea lui $f(n) \overset{(not)}{=} O(g(n))$

dacă $\exists\, n_0 \in N$ şi $c = const.$

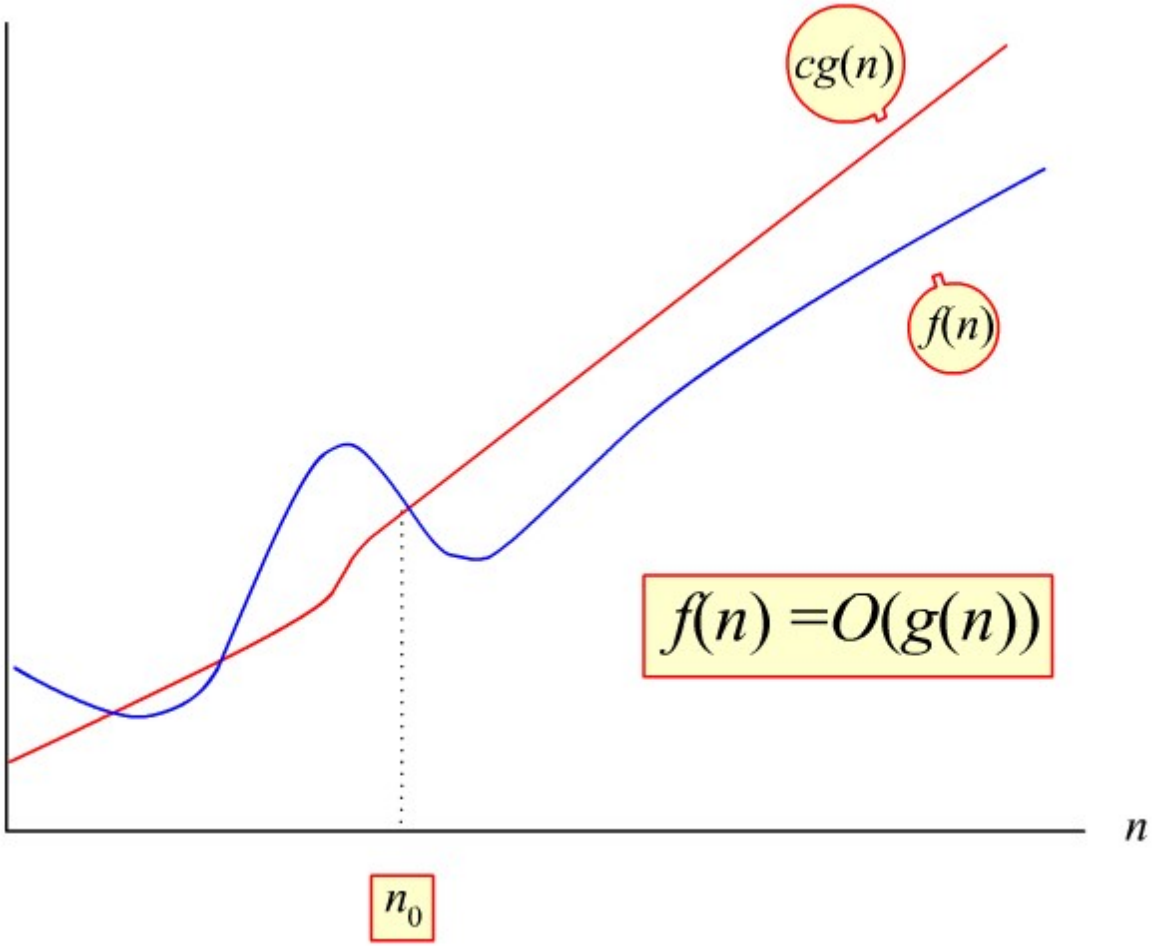
a.î. $\forall n \geq n_0, f(n) \leq cg(n).$

Fie $f(n) = 2n^3 + 10n^2 + 2, \forall n \geq 1$

$f(n) \leq 12n^3 \Rightarrow$ complex. $f(n) \overset{(not)}{=} O(n^3)$,

pt. că $\exists n_0 = 1$ şi $c = 12$ a.î. $\forall n \geq n_0, f(n) \leq cg(n).$

Notații asimptotice - O



$f(n), g(n) : \mathrm{N} \to R$, complexitatea lui $f(n) \overset{(not)}{=} O(g(n)) \overset{(not)}{=} g(n)$
dacă $\exists\, n_0 \in N$ şi $c = const.$ a.î. $\forall n \geq n_0, f(n) \leq cg(n).$

Notații asimptotice – clasificare algoritmi

$O(1)$ = constant

$O(\log n)$ = logaritmic

$O(n)$ = liniar

$O(n \log n)$ = supraliniar

$O(n^2)$ = patratic

$O(n^c)$ = polinomial

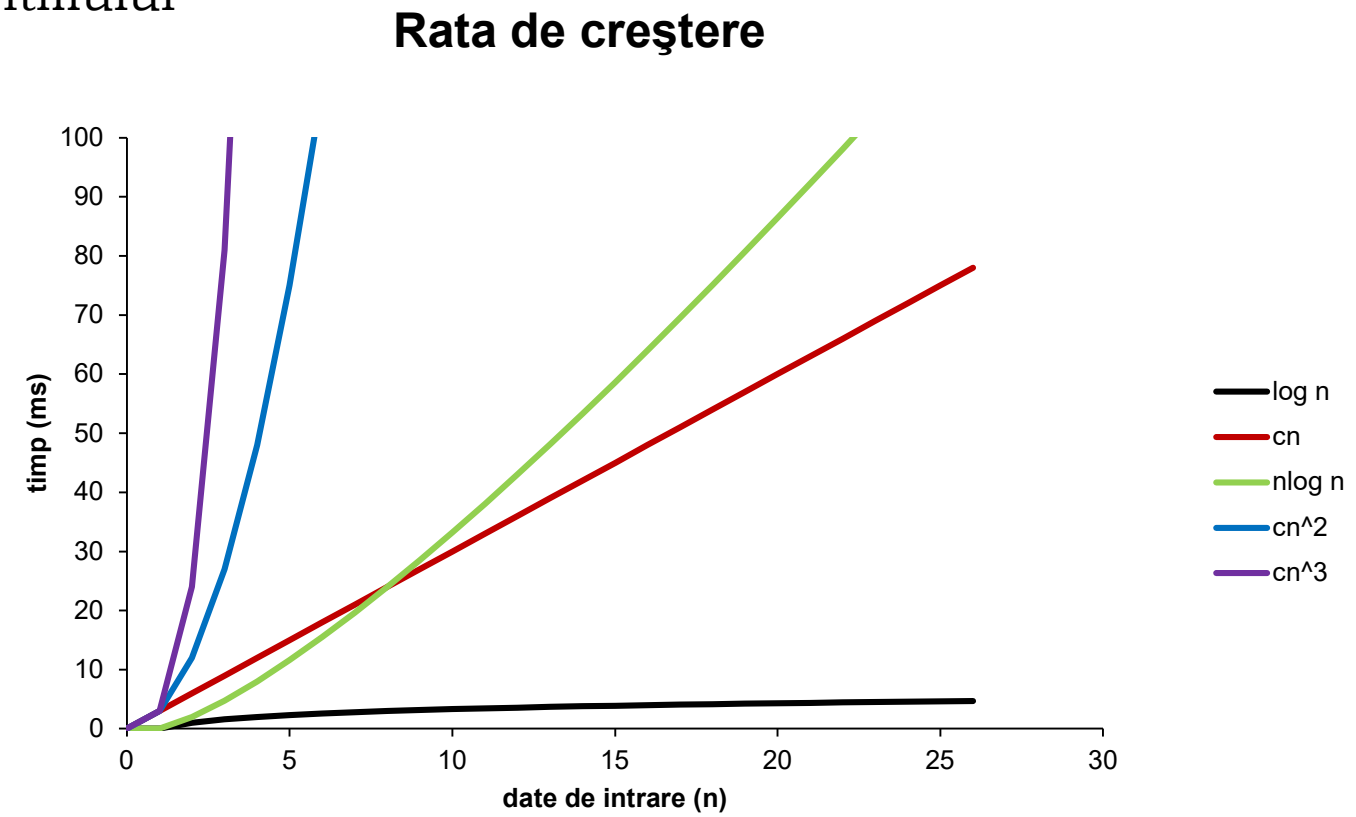
$O(c^n)$ = exponential

$O(n!)$ = factorial

c - constantă

Ordinul/rata de creştere a unui algoritm

- timpul de execuție a unui algoritm este o funcție de dimensiunea datelor de intrare și de complexitatea algoritmului



- $f(n)=n^3+n^2$
- $f(n)=n\log n+n+3$
- cu cât n creşte cu atât scade importanța termenilor de grad inferior

Ordinul/rata de creştere a unui algoritm

n	O($\log n$)	O(n)	O($n \log n$)	O(n^2)	O(n^3)	O (2^n)
$2^7 = 128$	$0.007\mu s$	$0.128\mu s$	$0.89\mu s$	$16\mu s$	$2ms$	$10^{31} ani$
$2^8 = 256$	$0.008\mu s$	$0.256\mu s$	$2\mu s$	$65\mu s$	$16ms$	$10^{69} ani$
$2^{10} = 1024$	$0.01\mu s$	$1\mu s$	$10\mu s$	$1ms$	1sec	$10^{300} ani$
$2^{12} = 4096$	$0.012\mu s$	$4\mu s$	$49\mu s$	$16ms$	68sec	$10^{1221} ani$
2^{20}	$0.02\mu s$	$1ms$	$20ms$	18.3min	$37 ani$	$10^{314565} ani$

Operaţii elementare: (i) adunare, scadere, înmulțire şi împărțire

(ii) comparații şi operatori logici

(iii) atribuiri

Notații asimptotice - Ω, Θ

Definiție: fie $f(n), g(n) : \mathrm{N} \to R$

$$f(n) \overset{(not)}{=} \Omega(g(n))$$

dacă $\exists\, n_0 \in N$ şi $c = \text{const}$. a.î. $\forall n \geq n_0$,

$f(n) \geq cg(n)$.

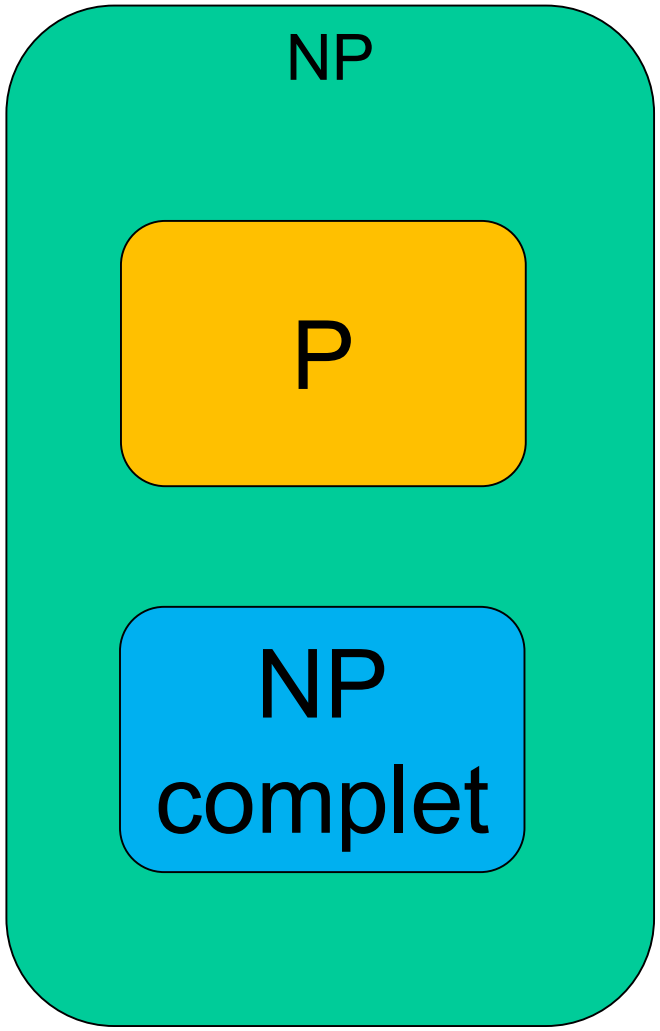
$$f(n) \overset{(not)}{=} \Theta(g(n))$$

dacă $\exists\, n_0 \in N$ şi $c_1, c_2 = \text{const}$. a.î. $\forall n \geq n_0$,

$c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Complexitate P/NP/NP-complet

- Algoritmi deterministici
- Algoritmi nondeterministici
- P = Polinomial
- NP = Nondeterministic Polinomial
- NP-complet



$P \neq NP$

Rezumat. Analiza algoritmilor

- Căutare liniară vs. Căutare binară
- Notații asimptotice - O
- Ordinul/rata de creștere a unui algoritm
- Tipuri de algoritmi

Tehnici de programare avansate

Operații pe biți

ovidiu.banias@upt.ro

Operații pe biți

Reprezentarea întregilor

Operanzi și operații pe biți în C

Sistemul binar de numerație

- Ce este un bit?

  O cifră în sistemul binar

- Ce valori poate stoca un bit?

  {0,1}

- De ce calculatoarele folosesc sistemul binar? De ce nu cel zecimal?

  Mai ușor de implementat hardware

Bit-ul

- Bit : {0,1}
- 4 biti : 0000 → 1111 – 2^4=16 valori distincte
- 8 biti : 00000000 → 11111111 – 2^8=256 valori distincte
- 16 biti : 2^16=65.535
- 24 biti : 2^24≈16.7 milioane
- 32 biti : 2^32 ≈ 4 Miliarde

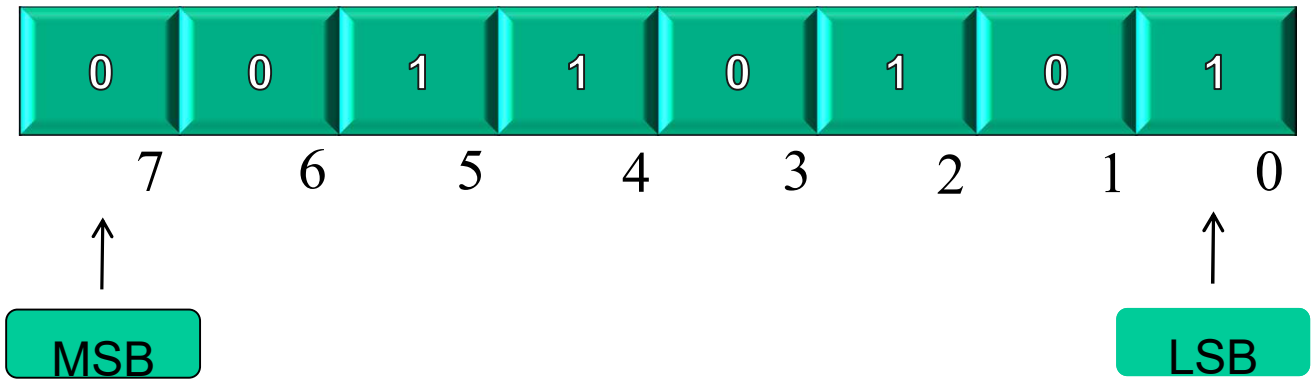
Octet-ul

- Ce este un octet (byte)?

1 octet = 8 biti

00000000 → 11111111

$$0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 53$$



Octet-ul

$X_{(10)}$	$X_{(2)}$
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
13	1101
21	00010101
253	11111101

- $X_{(10)} \rightarrow X_{(2)}$?
- $X_{(2)} \rightarrow X_{(10)}$?
- Cum se reprezintă în binar numerele negative?

Complement de 1

Complement de 2

Tipul de dată char / unsigned char (ASCII)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Exemple

$X_{(10)}$	$X_{(2)}$	$X_{(8)}$	$X_{(16)}$
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B

$X_{(10)}$	$X_{(2)}$	$X_{(8)}$	$X_{(16)}$
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
36	100100	44	24
61	111101	75	3D
79	1001111	117	4F
100	1100100	144	64
140	10001100	214	8C
254	11111110	376	FE
255	11111111	377	FF

Operatori pe biți

Operator	Denumire	Exemplu	Descriere
~	Not	$\sim v$	Inversează biţii var. ***v***
&	And	$v_1 \& v_2$	0&0=0, 0&1=0, 1&1=1
|	Or	$v_1 \mid v_2$	0|0=0, 0|1=1, 1|1=1
^	Xor (Sau exclusiv)	$v_1 \wedge v_2$	0^0=0, 0^1=1, 1^1=0
<<	Shift left	$v << i$	Deplasează var. ***v*** la stânga cu ***i*** *poziţii*
>>	Shift right	$v >> i$	Deplasare la dreapta cu ***i*** *poziţii*

Operații pe biți

v_1	v_2	$v_1 \,\&\, v_2$
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

v_1	v_2	$v_1 \mid v_2$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

v_1	v_2	$v_1 \,\hat{}\, v_2$
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

v	$\sim v$
0	**1**
1	**0**

v	i	$v << i$
0001	1	0000000**10**
0101	2	00010100
0001	3	0000**1000**
1001	4	**10010000**
0011	5	**01100000**

v	i	$v >> i$
00110110	1	00011011
01010011	2	00010100
00000011	3	00000000
00011001	4	0000000**1**
11111111	5	00000**111**

Măşti/Şabloane (Mask)

Măştile se folosesc pentru accesarea, setarea anumitor biți dintr-un octet/cuvânt. Fie **n** – octet şi **k={0,...,7}** – poziția unui bit în octet

SetFlag – setează bitul de pe poziția k la valoarea 1

$$n = n \mid (1 << k)$$

UnsetFlag – setează bitul de pe poziția k la valoarea 0

$$n = n \& \sim (1 << k)$$

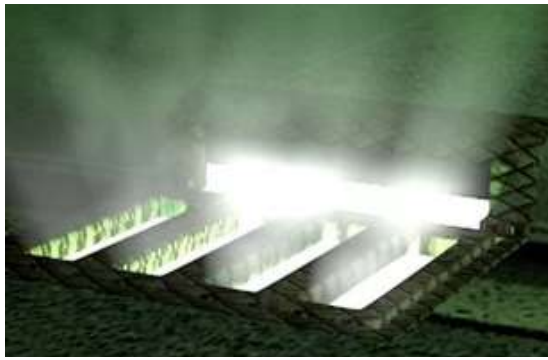
GetFlag – returnează valoarea bitului de pe poziția k

$$(n \,\&\, (1 << k) == (1 << k))$$

$$(1 \,\&\, (n >> k))$$

ChangeFlag – schimbă valoarea bitului de pe poziția k

$$n = n \hat{} (1 << k)$$



Probleme

1. Se dă o mulțime de numere naturale cu valori de la 0 la 1000. Se citesc de la intrare diferite valori în intervalul [0,1000], să se utilizeze un vector de octeți de dimensiune cât mai mică pentru memorarea elementelor mulțimii.

2. Fiind necesară setarea/verificarea permisiunilor pentru utilizatorii unei aplicații, și presupunând că informațiile legate de permisiune (per utilizator) pot fi salvate doar într-o variabilă de dimeniunea unui octet (limitări de memorie), să se implementeze funcții pentru setarea și verificarea permisiunilor unui anumit utilizator. Există 5 tipuri de permisiuni: Read/Write/Delete/Rename/Copy.

Tehnici de programare avansate

Alocarea dinamică a memoriei. Pointeri

ovidiu.banias@upt.ro

Cuprins

Pointeri. Utilitate

- Alocare dinamică a memoriei

  Compile time vs. Run time

  Optimizare

- Transmiterea parametrilor prin referință

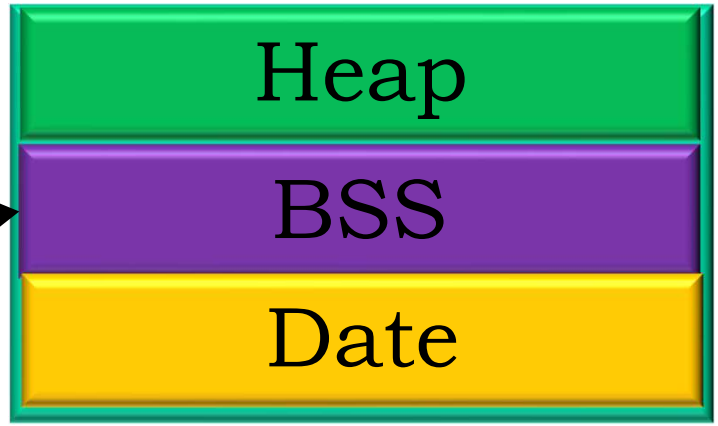
- Vectori și șiruri de caractere

Memoria

Segmentul de STIVĂ

- Variabile locale
- Apeluri de funcții

Segmentul de DATE



- Variabile globale, statice
- Alocare dinamică
- Variabile inițializate/neinițializate
- BSS= block starting with symbol

Segmentul de COD/TEXT

- Conține instrucțiunile executabile

Adrese de memorie

- Adresă = numărul de ordine al unui octet(cuvânt) în memorie
- Compilator pe 16/32/64 de biți
- O variabilă ocupă x octeți succesivi. Adresa primului octet=adresa variabilei

  char → 1 octet

  int → 2 octeți (sau 4)

- adresa unei variabile ≠ valoarea variabilei
- adresă = pointer

Variabile de tip pointer

- Sintaxa C:

```
tip    *numeVariabila
```

  - **tip** : *tip de dată standard* (char, int,...)

    sau *tip de dată compus*(struct, union,..)

  - ***** : operator de dereferențiere (indirectare)

  - **numeVariabila** : nume variabilă

```
tip    *numeVar
char   *adresa1;
int    *adresa2, *adresa3;
float  *adresa4;
```

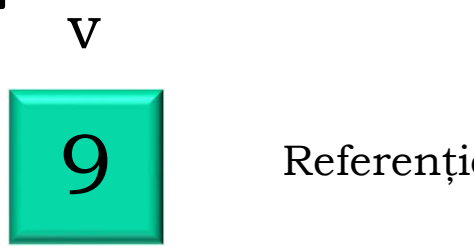
Pointer = variabilă ce are ca valoare o adresă de memorie

O variabilă de tip pointer, pointează către o anumită zonă din memorie (adresează o anumită zonă de memorie)

Variabile de tip pointer

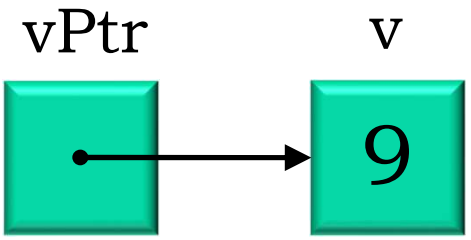
```
int v=7;

int *vPtr;

…

vPtr=&v;

…
```

Pointerul ***vPtr*** pointează către variabila ***v***



Referențiere directă

- ***&*** - operator de adresare (referențiere)
- ***0, NULL*** – inițializare pointer



Referențiere indirectă

Variabile de tip pointer. Dereferențiere

```
int v=9;
int *vPtr;

int main()
{
    printf("%p\n",vPtr);

    vPtr=&v;

    printf("%d\n",*vPtr);
    printf("%p\n",vPtr);
    printf("%p\n",&v);

    return 0;
}
```

***** - operator de indirectare

%p – specificator de formatare adresă de memorie

inițializare variabilă pointer

***vPtr** – valoarea stocată la adresa de memorie spre care pointează vPtr.

***** – operator de dereferențiere

&v – adresa de memorie la care este stocată valoarea variabilei v

& – operator de adresare

Operații cu pointeri. Exemplificare

```
int v=9;
int *vPtr, *vPtr2;

int main()
{
    vPtr=&v;

    *vPtr = *vPtr + 5; printf("%d\n",*vPtr);

    (*vPtr) ++; printf("%d\n",*vPtr);

    vPtr2=vPtr; (*vPtr) ++; printf("%d\n",*vPtr2);

    printf("%p\n",vPtr);
    printf("%p\n",vPtr2);

    return 0;
}
```

Alocarea dinamică a memoriei

```
#include <stdio.h>
#include <stdlib.h>

#define tip char

tip *vPtr, *vPtr2;

int main()
{
    vPtr=(tip *) malloc(sizeof(tip));
    vPtr2=(tip *) malloc(sizeof(tip));

    printf("%p\n",vPtr);
    printf("%p\n",vPtr2);
    printf("%p\n",vPtr2+1);

    printf("%d\n",(vPtr2-vPtr));
    return 0;
}
```

Transmiterea parametrilor prin referință

```
void Swap1(int p, int q){ // prin valoare
    int k;
    k=p;p=q;q=k;
}
void Swap2(int &p, int &q){ // prin referinta (C++)
    int k;
    k=p;p=q;q=k;
}
void Swap3(int *p, int *q){ // prin referinta
    int k;
    k=*p;*p=*q;*q=k;
}
int main(){
    a=2;b=5;
    Swap1(a,b);printf("Swap1 -> %d %d\n",a,b);
    Swap2(a,b);printf("Swap2 -> %d %d\n",a,b);
    Swap3(&a,&b);printf("Swap3 -> %d %d\n",a,b);
   return 0;
}
```

Vectori și pointeri

- Strânsă legătură între vectori şi pointeri
- Orice operație cu indecși şi vectori poate fi realizată cu pointeri