Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου

ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

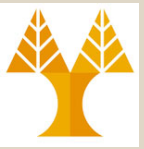
Διάλεξη 7: Συμβολοσειρές, Δείκτες και Παραδείγματα

(Κεφάλαιο 13, KNK-2ED)

Δημήτρης Ζεϊναλιπούρ

http://www.cs.ucy.ac.cy/courses/EPL232

Περιεχόμενο Διάλεξης 7



- **Συμβολοσειρές (Strings)**
  - Σταθερά (literal), Μεταβλητή (variable), Αρχικοποίηση, Ανάγνωση / Εκτύπωση
  - Παραδείγματα: Σάρωση, Μέτρηση Κενών Χαρακ.
- **Η Βιβλιοθήκη <String.h>**
  - Συναρτήσεις: strlen, strcpy, strcat, strcmp
  - Υλοποίηση Συναρτήσεων
- **Πίνακες Δεικτών (και Συμβολοσειρών)**
  - Ορίσματα Προγράμματος *argv[], argc
  - Παραδείγματα Επεξεργασίας

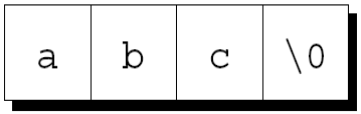
Σταθερά Συμβολοσειράς String Literal (or Constant)

- **Σταθερά Συμβολοσειράς (String Literal):** ακολουθία χαρακτήρων που εσωκλείεται από **δίπλα εισαγωγικά (")** και τερματίζεται αυτόματα από τον χαρακτήρα **NUL ('\0'):**

  `"Candy\nIs dandy\nBut liquor\n... --Ogden Nash\n"`

  – Το NUL ('\0') δεν δίνεται στον ορισμό της σταθεράς

  – Σεβόμενοι τον κανόνα 80 στηλών κατά την εκτύπωση …




```
printf("When you come to a fork in the road, take it. \
--Yogi Berra"); // ή printf("When you come to a fork in the road, take it.  "
                                    "--Yogi Berra");
```

- **`char *p; p = "abc"; /*** ΣΩΣΤΟ ***/`**

  – Το p δείχνει πάνω στον πρώτο χαρακτήρα της σταθεράς "abc".

  – Ορθός ορισμός, εφόσον το `"abc"` είναι ήδη στην μνήμη (αποθηκεύεται στο Data Segment και όχι στο Program Stack)

  – `char *p; p[1]= 'a'; /*** ΛΑΘΟΣ- p δεν δεσμεύτηκε χώρος***/`

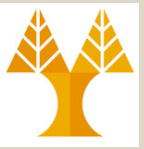
Σταθερά Χαρακτήρα

- **`char *p = "abc"; *p = 'd'; /*** ΛΑΘΟΣ - Δεν επιτρέπεται να μεταβάλλεται μια σταθερά (το “abc”) ***/`**

Συμβολοσειρά (Ο Χαρακτήρας NUL \0)

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		Spa	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;	DE

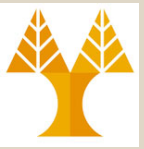
Σταθερά Συμβολοσειράς και Σταθερά Χαρακτήρα



- Μια **σταθερά συμβολοσειράς (string literal)** η οποία περιέχει ένα χαρακτήρα **ΔΕΝ είναι το ίδιο** με μια **σταθερά χαρακτήρα (character constant).**
  - `"a"` αναπαριστάται από ένα δείκτη σε char (τερματίζεται με NUL καταλαμβάνοντας 2 bytes) .
  - `'a'` αναπαριστάται από ένα χαρακτήρα (και καταλαμβάνοντας 1 byte)
- Επομένως:

```
printf("\n");    /*** ΣΩΣΤΟ ***/
printf('\n');    /*** ΛΑΘΟΣ ***/
```

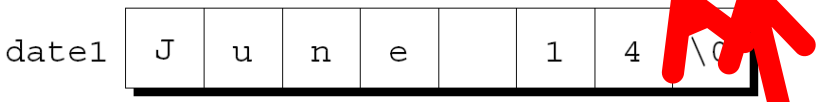
Μεταβλητές Συμβολοσειράς (String Variables)



- **Μεταβλητή Συμβολοσειράς:** Μια μεταβλητή που μπορεί να αναπαραστήσει μια συμβολοσειρά.
  - Για string μήκους N, είναι ένας πίνακας N+1 θέσεων.
  - `#define STR_LEN 80`
  - `char str[STR_LEN+1];`

- **Σταθερά Συμβολοσειράς**

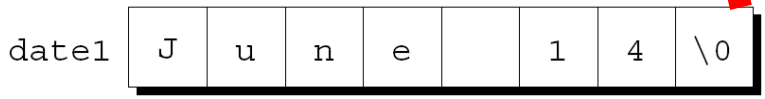
```
char *date1 = "June 14";
```



date1 **τερματίζεται με NUL '\0'**

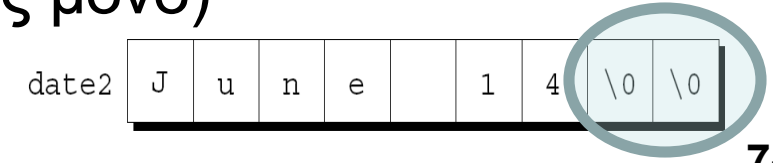
- **Μεταβλητή Συμβολοσειράς**

```
char date1[8] = "June 14";
```



- Επιπλέον NUL προστίθενται από τον μεταγλωττιστή εάν υπάρχει χώρος (για μεταβλητές μόνο)

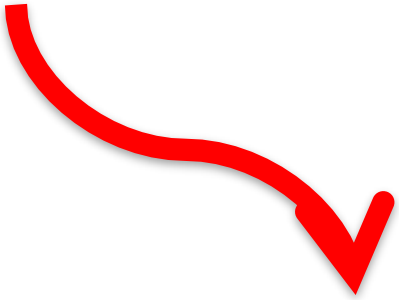
```
char date2[9] = "June 14";
```



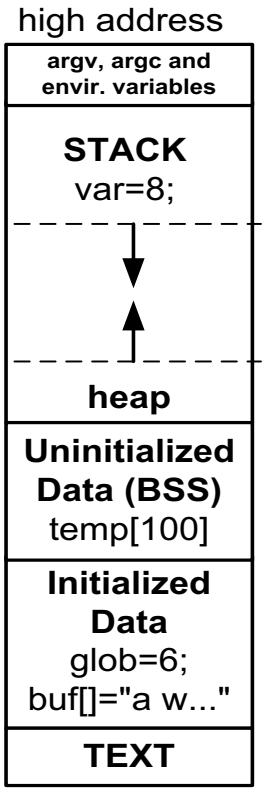
Σταθερά & Μεταβλητή Συμβολοσειράς

ΠΡΟΓΡΑΜΜΑ

```
char date1[8] = "June 14";
char *date1;
```



"June 14";



Μεταβλητές Συμβολοσειράς (Αρχικοποίηση)

literal — Δηλώνονται αυτοί οι ορισμοί (declarations) — variable

```
const char *msg="Hello"; ή const char msg[]="Hello";
```

αλλά υπάρχουν και άλλοι τρόποι…

Σωστό αλλά άκομψο

```
char msg[6];
msg[0] = 'H';
msg[1] = 'e';
msg[2] = 'l';
msg[3] = 'l';
msg[4] = 'o';
msg[5] = '\0';
```

Σωστό αλλά άκομψο

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

Σωστό αλλά άκομψο

```
char msg[6]={'H','e','l','l','o','\0'};
```

Σωστό αλλά σπάταλο

```
char msg[40]="Hello";
```

Λάθος (ξεχνάμε το \0)

```
char msg[ ]={'H','e','l','l','o'};
```

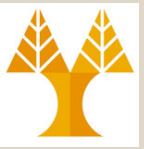
Λάθος (το \0 δεν προστίθεται έλλειψη χώρου)

```
char msg[5]={'H','e','l','l','o','\0'};
```

Λάθος (δεν δεσμεύεται χώρος) – ΕΠΙΚΙΝΔΥΝΟ (συχνό λάθος)

```
char *p; p[0] = 'a'; /*** ΛΑΘΟΣ ***/
```

Ανάγνωση/Εκτύπωση String



scanf ("%s", name)

ΠΡΟΣΟΧΗ: Δεν χρησιμοποιείτε το &, γιατί το name είναι πίνακας. Θυμηθείτε ότι σε άλλους τύπους δεδομένων χρησιμοποιείται το & π.χ. **scanf("%d", &i);**

- **Για εισαγωγή συμβολοσειράς με κενά χρησιμοποιείται η**
  - fgets(name, sizeof(name), stdin); ή
  - #define MAX "50"

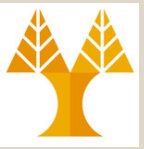
    scanf("%" MAX "[^\n]", name);
- **Για σάρωση προτύπου**
  - sscanf (sentence,"%s %d", str, &i);

printf ("%s", name) Για εκτύπωση συμβολοσειρών

sprintf (buffer, "(%d,%d,%d)", a, b, c);

- Δημιουργία Προτύπου
- Όπου buffer είναι δεσμευμένο χώρος, π.χ., char buffer[50];

Παράδειγμα (Σάρωση Συμβολοσειρών)



- Γράψετε μια ασφαλή συνάρτηση σάρωσης εισόδου στη γλώσσα C, η οποία να διαβάζει το περιεχόμενο της εισόδου σε μεταβλητή πίνακα `str` με τις εξής συνθήκες:
  - (1) Δεν διαγράφει τους white-space χαρακτήρες
  - (2) Σταματά μόλις διαβάσει τον πρώτο χαρακ. γραμμής `\n` (που δεν ανήκει στο str) ή εάν έχει διαβάσει `n` χαρακτήρες
  - (3) Αγνοεί οποιουσδήποτε χαρακτήρες μετά το `\n`
  - (4) Επιστρέφει τον αριθμό των χαρακτήρων που αναγνώστηκαν
- Πρότυπο Συνάρτησης:

```
int read_line(char str[], int n);
```

Παράδειγμα (Σάρωση Συμβολοσειρών)

```
int read_line(char str[], int n)  {

  int ch, i = 0;

  while ((ch = getchar()) != '\n') {
    if (i < n) {
      str[i] = ch;
       i++;
    }
  }
  str[i] = '\0';    /* terminates string */
  return i;         /* # of chars stored */
}
```

- Το `ch` έχει τύπο `int` παρά `char` εφόσον `getchar` επιστρέφει ακέραια τιμή `int`.

Παράδειγμα (Μέτρηση Κενών σε Συμβολοσειρά)

Γράψετε μια στη γλώσσα C, η οποία μετρά και επιστρέφει τον αριθμό των spaces σε μια συμβολοσειρά εισόδου `str`

Πρότυπο Συνάρτησης:

```
int count_spaces(const char s[])
```

```
int count_spaces(const char s[])
{
  int count = 0, i;

  for (i = 0; s[i] != '\0'; i++){
    if (s[i] == ' ') {
      count++;
    }
  }
  return count;
}
```

Θα μπορούσαν να αντιστραφούν

```
int count_spaces(const char *s){
  int count = 0;

  for (; *s != '\0'; s++) {
    if (*s == ' ') {
      count++;
    }
  }
  return count;
}
```

Λύση με αριθμητική δεικτών

Η Βιβλιοθήκη <string.h> (C String Library)

- Εφόσον τα Strings είναι πίνακες χαρακτήρων στη C, αυτά **ΔΕΝ μπορούν** να **αντιγραφούν**, **συγκριθούν**, κτλ με **απλούς τελεστές** (π.χ., =, ==, κτλ) που μάθαμε για τους άλλους τύπους.
- **Συνηθισμένο λάθος:**

```
typedef unsigned long int ADDR;
printf("%lx %lx", (ADDR)str1, (ADDR)str2);
7ffffa0f 7ffffa0e
```

```
char str1[10], str2[10];
str1 = "abc";   /*** ΛΑΘΟΣ μεταγλώττισης ***/
str2 = str1;    /*** ΛΑΘΟΣ μεταγλώττισης ***/
if (str1 == str2) ...    /*** ΟΧΙ αναμενόμενο αποτέλεσμα,
συγκρίνει τις διευθύνσεις των str1,str2 επιστρέφοντας FALSE 0) ***/
```

- Ορθό (πρόκειται για ορισμό – declaration - όχι ανάθεση)

```
char str1[10] = "abc";      // OK
```

- Ορθό: `if (str1[0] == str2[0])      // OK`

Η Βιβλιοθήκη <string.h> (C String Library)

- Η Βιβλιοθήκη <string.h> περιέχει ένα μεγάλο σύνολο λειτουργιών επεξεργασίας συμβολοσειρών.

  #include <string.h>(strcpy,strcat,strlen,strcmp,..)

- **Αντιγραφή (Copy)** του string s2 στο s1 (return: διεύθυνση του s1)

```
char *strcpy(char *s1, const char *s2);
```

  – Ασφαλής Αντιγραφή του `str2` στο `str1` (εάν s2 μεγαλύτερο s1):

```
strncpy(str1, str2, sizeof(str1) - 1); str1[sizeof(str1)-1] = '\0';
```

- **Εύρεση Μήκους (Length)** συμβολοσειράς (δεν μετρά το '\0')

```
size_t strlen(const char *s); // sizeof περιλαμβάνει '\0' και πέρα (π.χ., εάν s[10] = "abc")
```

  – `size_t` είναι ένα `typedef` όνομα της C για unsigned integer τύπους.

- **Επικόλληση (Append) S2 στο S1:**

```
char *strcat(char *s1, const char *s2);
ΠΡΟΣΟΧΗ: char str1[6] = "abc"; strcat(str1, "def"); /** ΕΠΙΚΙΝΔΥΝΟ ***/
ΑΣΦΑΛΗΣ: strncat(str1, str2, sizeof(str1) - strlen(str1) - 1);
```

Εάν str1[100] δίνει 100

δίνει 3

Διαθέσιμος χώρος στο str1

Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strlen`)

- Για εξάσκηση, θα υλοποιήσουμε τώρα κάποιες από τις συναρτήσεις της `string.h`.
  - Το βιβλίο περιέχει διαφορετικές εκδόσεις των υλοποιήσεων για κάθε μια από τις συναρτήσεις.
- Η συνάρτηση `strlen(s1)` μετρά το μήκος μιας συμβολοσειράς (χωρίς το NUL)

Έκδοση Α

Ίδιο με (*s != 0) ή (*s) // 1 TRUE

```
size_t mystrlen(const char *s) {
    size_t n;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}
```

Έκδοση Β

```
size_t mystrlen(const char *s) {
    size_t n = 0;
    while (*s != '\0') {
      s++; n++;
    }
    return n;
}
```

Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strcat`)

- Η συνάρτηση `strcat(s1, s2)` αντιγράφει το `s2` στο τέλος του `s1`, π.χ.,

```
int main() {
    char ma[10]="HELLO";
    char mb[10]="CAT";
    strcat(ma,mb);
    return 0;
}
```

Πρίν

	0	1	2	3	4	5	6	7	8	9
s1	H	E	L	L	O	\0	?	?	?	?
s2	C	A	T	\0	?	?	?	?	?	?

Μετά

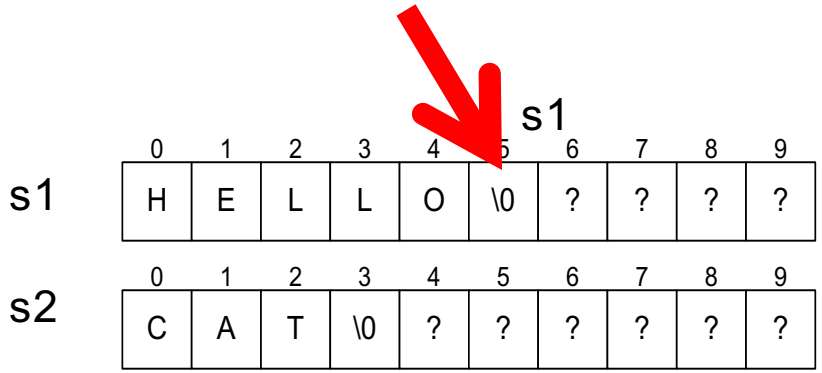
	0	1	2	3	4	5	6	7	8	9
s1	H	E	L	L	O	C	A	T	\0	?
s2	C	A	T	\0	?	?	?	?	?	?

Αλγόριθμος

- Βρές τον χαρακτήρα NUL στο τέλος του `s1` και κάνε τον δείκτη `p` να δείχνει στο σημείο αυτό.
- Αντίγραψε ένα-ένα τους χαρακτήρες από το `s2` ξεκινώντας από το σημείο που δείχνει το `p`.
- Δεν μελετούμε ακόμη το θέμα της υπερχείλισης του s1.

Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strcat`)

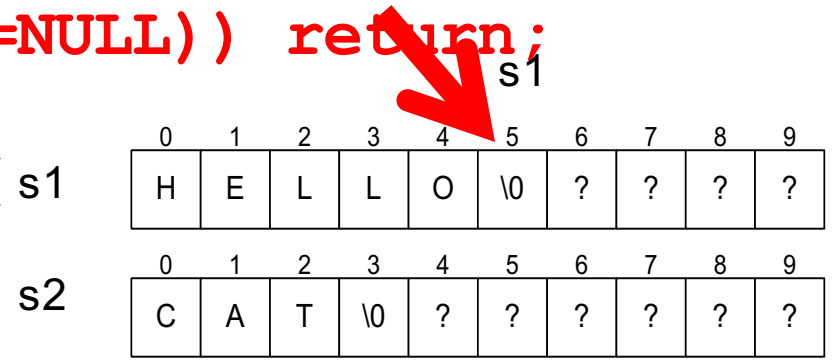
```
void mystrcat(char *s1, const char *s2) {
    if ((s1==NULL)||(s2==NULL)) return;
    while (*s1 != '\0') { // εύρεση NUL
       s1++;
    }
    while (*s2 != '\0') { // Αντιγραφή s2 -> s1
       *s1 = *s2;
       s1++; s2++;
    }
    *s1 = '\0';
    return;
}
```



Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strcat`)

- Μια πιο συμπαγής έκδοση της `strcat`:

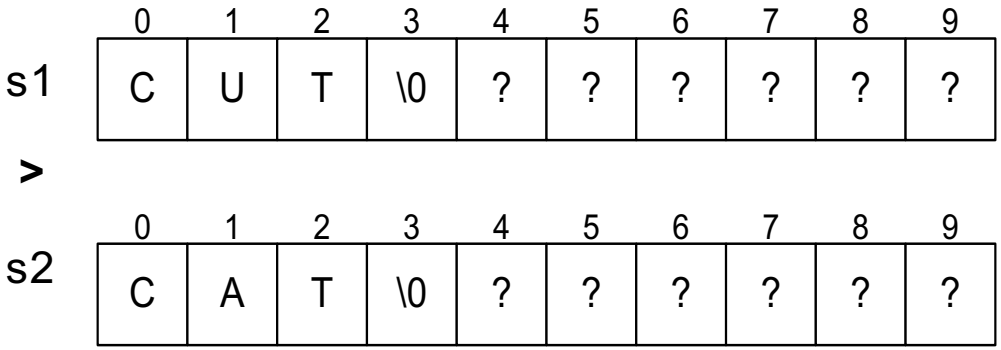
```
void strcat(char *s1, const char *s2)
{
  if ((s1==NULL)||(s2==NULL)) return;

  while (*s1 != '\0') {
    s1++;
  }
  while (*s1 = *s2) { // ανάθεση, όχι σύγκριση
    s1++; s2++;
  }
  return;
}
```



Όταν το `s2` φτάσει το `'\0'` τότε γίνεται `*s1=0`, δηλ., `(*s1=0)` γίνεται 0 (`FALSE`) και διακόπτεται το loop

Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strcmp`)

- Γράψετε συνάρτηση που συγκρίνει τα **αλφαριθμητικά** s1 και s2 και επιστρέφει
  - **1:** αν το s1 είναι μεγαλύτερο λεξικογραφικά
  - **0:** αν είναι ίσα (ή το ένα είναι υπο-συμβολοσειρά του άλλου) και
  - **-1:** αν το s2 είναι μεγαλύτερο λεξικογραφικά

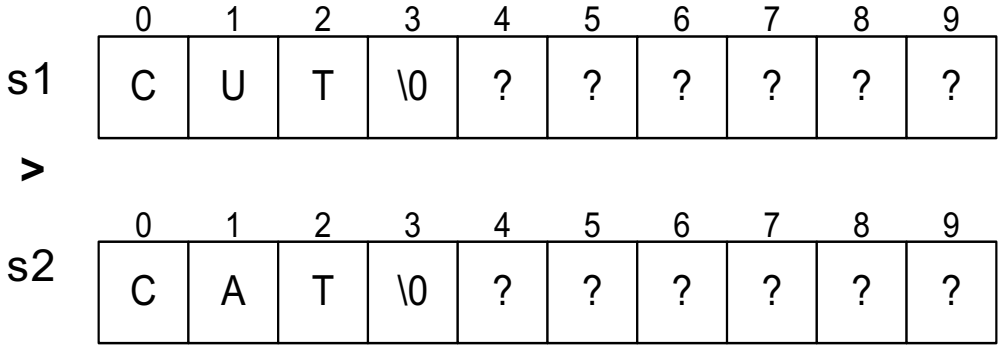


Επιστρέφει 1

Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strcmp`)

- **Εκδοχή Α (P.J. Plauger, Standard C Library):**

```
int mystrcmp(const char *s1, const char *s2) {
    int i;
    for(i=0; s1[i] == s2[i]; i++) {
        if(s1[i] == 0) // Εάν φτάσαμε τον '\0' χαρακτήρα
            return 0;
    } // χαρ. του s1 δεν είναι ίσος με τον αντίστοιχο του s2
    return (s1[i] < s2[i]) ? -1 : 1;
}
```



Υλοποίηση Συναρτήσεων (Η Συνάρτηση `strcmp`)

- **Β) Εκδοχή Β (μόνο δείκτες)**

```
int mystrcmp(const char *s1, const char *s2) {
    if ((s1==NULL) && (s2!=NULL)) return -1;
    else if ((s1!=NULL) && (s2==NULL)) return 1;
    else if ((s1==NULL) && (s2==NULL)) return 0;

    for(; *s1 == *s2; ++s1, ++s2)
         if(*s1 == 0)
             return 0;

     return (*s1 < *s2) ? -1 : 1;
}
```

Πίνακες Συμβολοσειρών (Arrays of Strings)

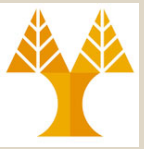
- Υπάρχουν πολλοί τρόποι να αποθηκεύσουμε ένα πίνακα από strings.
- Μια προσέγγιση είναι να χρησιμοποιήσουμε ένα **2-d πίνακα χαρακτήρων** (ένα string ανά γραμμή):

```
char planets[][8] = {
    "Mercury", "Venus", "Earth",
    "Mars", "Jupiter", "Saturn",
    "Uranus", "Neptune", "Pluto"
};
```

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0
5	S	a	t	u	r	n	\0	\0
6	U	r	a	n	u	s	\0	\0
7	N	e	p	t	u	n	e	\0
8	P	l	u	t	o	\0	\0	\0

Σπάταλη Προσέγγιση!

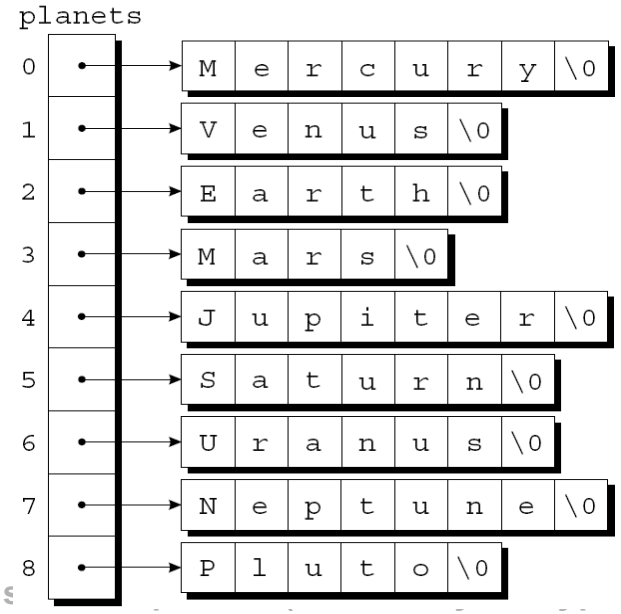
Πίνακες Δεικτών (Arrays of Pointers)



- Μια καλύτερη προσέγγιση είναι η δημιουργία ενός ***ακανόνιστου πίνακα (ragged array),*** των οποίων οι γραμμές έχουν διαφορετικά μεγέθη.
- Κάτι τέτοιο μπορεί να επιτευχθεί με ένα **πίνακα δεικτών (array of pointers) :**

```
char *planets[] = {
    "Mercury", "Venus", "Earth",
    "Mars", "Jupiter", "Saturn",
    "Uranus", "Neptune","Pluto"
};
```

Αποδοτική Προσέγγιση!



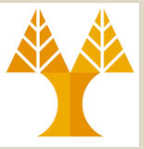
Πολυδιάστατοι Πίνακες vs. Πίνακες Δεικτών

- Έστω

```
int a[10][20]
int *b[10];
```

- **Ποια η διαφορά ανάμεσα στις δυο δηλώσεις;**
  - ο `a` είναι πραγματικά δισδιάστατος πίνακας: κατά τον ορισμό του δεσμεύθηκαν `200` συνεχόμενες θέσεις.
  - Κατά τον ορισμό του `b` κατανέμεται χώρος για `10` δείκτες. Απόδοση αρχικών τιμών πρέπει να **γίνει ρητά** είτε **στατικά** ή με κώδικα (malloc() που θα δούμε αργότερα)
- Πλεονέκτημα ενός **πίνακα με δείκτες** είναι ότι κάθε δείκτης μπορεί να δείχνει σε γραμμή με διαφορετικό μήκος

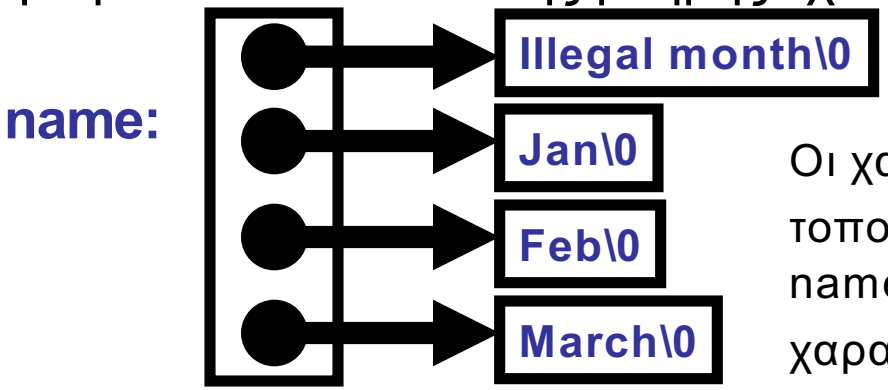
Παράδειγμα



Έστω

```
char  *name[]       = {"Illegal month", "Jan", "Feb", "March"}
char  aname[] [15] = {"Illegal month", "Jan", "Feb", "March"}
```

Γραφικά στο επίπεδο της μνήμης έχουμε



Οι χαρακτήρες κάθε συμβολοσειράς τοποθετούνται κάπου στη μνήμη και στο name[i] τοποθετείται δείκτης σ'αυτούς τους χαρακτήρες.

Ορίσματα Προγράμματος (Command-Line Arguments)

- Μέχρι τώρα ορίζαμε τη συνάρτηση **main** με παράμετρο void θέλοντας να δείξουμε ότι η συνάρτηση **main** δε δέχεται ορίσματα.

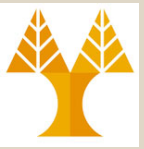
```
int main(void) {
    . . .
}
```

- Αυτό όμως δε σημαίνει ότι δεν μπορούμε να περάσουμε ορίσματα. Τα δυνατά ορίσματα όμως της **main** είναι καθορισμένα και είναι τα εξής:

```
int main(int argc, char *argv[]) {
    . . .
}
```

- Τα ορίσματα περνούνται στο πρόγραμμα από τη γραμμή εντολών τη στιγμή που αρχίζει η εκτέλεσή του ➔ **ορίσματα στη γραμμή εκτέλεσης**.

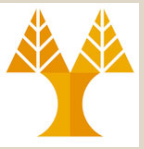
Ορίσματα Προγράμματος (Command-Line Arguments)



- Το πρώτο όρισμα **argc**, το οποίο είναι τύπου ακέραιος, είναι ο αριθμός των ορισμάτων της γραμμής διαταγών, με τα οποία έχει κληθεί το πρόγραμμα
  - συμπεριλαμβανομένου και του ονόματος του εκτελέσιμου αρχείου).

- Το δεύτερο όρισμα **argv** είναι δείκτης για έναν πίνακα συμβολοσειρών ο οποίος περιέχει τα ορίσματα. Κατά σύμβαση **(πίνακας δεικτών)**
  - **argv[0]** είναι το όνομα με το οποίο κλήθηκε το πρόγραμμα.
  - **argv[1], …, argv[argc - 1],** είναι τα υπόλοιπα ορίσματα με την σειρά που δόθηκαν στη γραμμή εντολής.
  - **argv[argc]** περιέχει το μηδενικό δείκτη (κενή συμβολοσειρά: “”).

Ορίσματα Προγράμματος (Command-Line Arguments)



- Παράδειγμα: Έστω ένα πρόγραμμα C το οποίο έχει τη μορφή:

```
int main(int argc, char *argv[]) {
        ......
}
```

- Θεωρείστε επίσης ότι το εκτελέσιμο αρχείο του παραπάνω προγράμματος έχει ονομαστεί **prog**. Τότε κατά την κλήση του **prog** υπό τη μορφή:

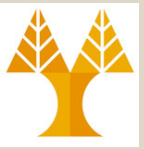
$ prog opt1 opt2 opt3

έχουμε την εξής ανάθεση τιμών στα ορίσματα της main:

argc = 4

argv[0] = “prog”
argv[1] = “opt1”
argv[2] = “opt2”
argv[3] = “opt3”
argv[4] = NULL

Παράδειγμα



- Ζητούμενο: πρόγραμμα που κατά την κλήση του με n ορίσματα (συμπεριλαμβανομένου και του εκτελέσιμου αρχείου) αντηχεί τα n-1 τελευταία στην οθόνη. Για παράδειγμα αν το πρόγραμμα αυτό λέγεται **echo**, τότε μία κλήση της μορφής:

$ echo Hello world!

θα εμφάνιζε στην έξοδο

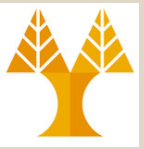
$ Hello world!

- Οι τυπικές παράμετροι της συνάρτησης main θα έχουν τιμές:

argc = 3

argv[0] = “echo”
argv[1] = “Hello”
argv[2] = “world!”
argv[3] = “”

Παράδειγμα



```
#include <stdio.h>

int main (int argc, char *argv[]){

    int i;

    for(i = 1; i < argc; i++)
       printf("%s%s", argv[i],
                    (i < argc - 1) ? " " : "");
    printf("\n");
    return 0;
}
```