Διάλεξη 14 - Εμβέλεια, Μνήμη και Συμβολοσειρές

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινίσεις

- Πιθανή ημερομηνία αναπλήρωσης: Πέμπτη 28/11, 7μμ-9μμ
    - Θα σταλεί και ανακοίνωση στο piazza

Σήμερα

- Κατηγορίες Μνήμης
- Διαχείριση Μνήμης
- Συναρτήσεις, Μεταβλητές,

  Δείκτες, Πίνακες
- Παραδείγματα

Σήμερα

- Εμβέλεια Μεταβλητών
- Παγκόσμια / Στατική Μνήμη
- Συμβολοσειρές (strings)

Εμβέλεια Μεταβλητής (Variable Scope)

Εμβέλεια μιας μεταβλητής λέγεται το μέρος του προγράμματος που η μεταβλητή είναι ορατή / προσβάσιμη.

Το σημείο δήλωσης μιας μεταβλητής στο πρόγραμμα καθορίζει την εμβέλειά της.

Δύο είδη μεταβλητών και εμβέλειας:

1. Τοπικές μεταβλητές (local variables)
2. Παγκόσμιες μεταβλητές (global variables)

Τοπικές Μεταβλητές (Local Variables)

Μια τοπική μεταβλητή (local variable) δηλώνεται μέσα στο σώμα μιας συνάρτησης. Η εμβέλειά της είναι από το σημείο δήλωσής της μέχρι το τέλος του block εντολών μέσα στο οποίο ορίστηκε. Αποθηκεύονται στην στοίβα.

```
void foo(int bar) {

  printf("%d\n", bar);

  int i = 42;

  printf("%d\n", i);

  int j;

  for(j = 0; j < i; j++)

     printf("%d %d\n", bar, i * j);

}
```

Scope της μεταβλητής ορίσματος bar (αρχικοποιείται κατά την κλήση της foo)

Τοπικές Μεταβλητές (Local Variables)

Μια τοπική μεταβλητή (local variable) δηλώνεται μέσα στο σώμα μιας συνάρτησης. Η εμβέλειά της είναι από το σημείο δήλωσής της μέχρι το τέλος του block εντολών μέσα στο οποίο ορίστηκε. Αποθηκεύονται στην στοίβα.

```
void foo(int bar) {

  printf("%d\n", bar);

  int i = 42;

  printf("%d\n", i);

  int j;

  for(j = 0; j < i; j++)

     printf("%d %d\n", bar, i * j);

}
```

Scope της τοπικής μεταβλητής i

Τοπικές Μεταβλητές (Local Variables)

Μια τοπική μεταβλητή (local variable) δηλώνεται μέσα στο σώμα μιας συνάρτησης. Η εμβέλειά της είναι από το σημείο δήλωσής της μέχρι το τέλος του block εντολών μέσα στο οποίο ορίστηκε. Αποθηκεύονται στην στοίβα.

```
void foo(int bar) {
  printf("%d\n", bar);
  int i = 42;
  printf("%d\n", i);
  int j;
  for(j = 0; j < i; j++)
     printf("%d %d\n", bar, i * j);
}
```

Scope της μεταβλητής ορίσματος j

Τοπικές Μεταβλητές (Local Variables)

Η τοπική μεταβλητή μιας συνάρτησης δεν είναι ορατή σε κάποια άλλη συνάρτηση και επομένως μπορούμε να δηλώσουμε μεταβλητές με το ίδιο όνομα σε άλλες συναρτήσεις και να αναφέρονται σε άλλες θέσεις μνήμης.

```
void foo() {
  int i = 43;
}
int main() {
  int i = 42;
  foo();
  return i;
}
```

Τι θα επιστρέψει αυτό το πρόγραμμα;

Τοπικές Μεταβλητές (Local Variables)

Η τοπική μεταβλητή μιας συνάρτησης δεν είναι ορατή σε κάποια άλλη συνάρτηση και επομένως μπορούμε να δηλώσουμε μεταβλητές με το ίδιο όνομα σε άλλες συναρτήσεις και να αναφέρονται σε άλλες θέσεις μνήμης.

```
void foo() {
  int i = 43;
}
int main() {
  int i = 42;
  foo();
  return i;
}
```

Τι θα επιστρέψει αυτό το πρόγραμμα;

```
$ gcc -o local local.c
$ ./local
$ echo $?
42
```

Τοπικές Μεταβλητές (Local Variables)

Η τοπική μεταβλητή μιας συνάρτησης δεν είναι ορατή σε κάποια άλλη συνάρτηση και επομένως μπορούμε να δηλώσουμε μεταβλητές με το ίδιο όνομα σε άλλες συναρτήσεις και να αναφέρονται σε άλλες θέσεις μνήμης.

```
void foo(int i) {
  i++;
}
int main() {
  int i = 42;
  foo(i);
  return i;
}
```

Τι θα επιστρέψει αυτό το πρόγραμμα;

Τοπικές Μεταβλητές (Local Variables)

Η τοπική μεταβλητή μιας συνάρτησης δεν είναι ορατή σε κάποια άλλη συνάρτηση και επομένως μπορούμε να δηλώσουμε μεταβλητές με το ίδιο όνομα σε άλλες συναρτήσεις και να αναφέρονται σε άλλες θέσεις μνήμης.

```
void foo(int i) {
  i++;
}
int main() {
  int i = 42;
  foo(i);
  return i;
}
```

Τι θα επιστρέψει αυτό το πρόγραμμα;

```
$ gcc -o local2 local2.c
$ ./local2
$ echo $?
42
```

Παγκόσμιες Μεταβλητές (Global Variables)

Μια παγκόσμια μεταβλητή (global variable) δηλώνεται έξω από οποιαδήποτε συνάρτηση. Η εμβέλειά της είναι από το σημείο δήλωσής της μέχρι το τέλος του αρχείου μέσα στο οποίο ορίστηκε.

```
int baz = 42;

void foo() {

  printf("baz: %d\n", baz);

}

void bar() {

  baz++;

}
```

Scope της μεταβλητής ορίσματος baz - ορατή και στην foo και στην bar

Επισκίαση Μεταβλητής (Variable Shadowing)

Όταν μια δήλωση μεταβλητής βρίσκεται εντός εμβέλειας άλλης μεταβλητής με το ίδιο όνομα, τότε η εσωτερική μεταβλητή επισκιάζει (shadows) την άλλη.

```c
#include <stdio.h>

int baz = 42;

int main() {
  int baz = 43;
  printf("baz: %d\n", baz);
  return 0;
}
```

Επισκίαση Μεταβλητής (Variable Shadowing)

Όταν μια δήλωση μεταβλητής βρίσκεται εντός εμβέλειας άλλης μεταβλητής με το ίδιο όνομα, τότε η εσωτερική μεταβλητή επισκιάζει (shadows) την άλλη.

```c
#include <stdio.h>

int baz = 42;

int main() {
  int baz = 43;
  printf("baz: %d\n", baz);
  return 0;
}
```

Scope της παγκόσμιας μεταβλητής baz

Scope της τοπικής μεταβλητής baz

Τι θα τυπώσει;

Επισκίαση Μεταβλητής (Variable Shadowing)

Όταν μια δήλωση μεταβλητής βρίσκεται εντός εμβέλειας άλλης μεταβλητής με το ίδιο όνομα, τότε η εσωτερική μεταβλητή επισκιάζει (shadows) την άλλη.

```
#include <stdio.h>

int baz = 42;

int main() {
  int baz = 43;
  printf("baz: %d\n", baz);
  return 0;
}
```

Scope της παγκόσμιας μεταβλητής baz

Scope της τοπικής μεταβλητής baz

Τι θα τυπώσει;

```
$ ./shadow
baz: 43
```

Στατικές Μεταβλητές (Static Variables)

Μια μεταβλητή λέγεται στατική (static), όταν διατηρεί την τιμή της ανάμεσα σε κλήσεις συναρτήσεων μέχρι το τέλος του προγράμματος. Οι στατικές μεταβλητές αρχικοποιούνται μόνο στην πρώτη κλήση της συνάρτησης.

```c
#include <stdio.h>

void foo() {

  static int counter = 0; int i = 0;

  counter++; i++;

  printf("%d %d\n", counter, i);

}

int main() {

  foo(); foo(); foo();

  return 0;

}
```

Τι θα τυπώσει;

Στατικές Μεταβλητές (Static Variables)

Μια μεταβλητή λέγεται στατική (static), όταν διατηρεί την τιμή της ανάμεσα σε κλήσεις συναρτήσεων μέχρι το τέλος του προγράμματος. Οι στατικές μεταβλητές αρχικοποιούνται μόνο στην πρώτη κλήση της συνάρτησης.

```c
#include <stdio.h>

void foo() {

  static int counter = 0; int i = 0;

  counter++; i++;

  printf("%d %d\n", counter, i);

}

int main() {

  foo(); foo(); foo();

  return 0;

}
```

Τι θα τυπώσει;

```
$ ./static

1 1

2 1

3 1
```

Στατικές Μεταβλητές (Static Variables)

Μια μεταβλητή λέγεται στατική (static), όταν διατηρεί την τιμή της ανάμεσα σε κλήσεις συναρτήσεων μέχρι το τέλος του προγράμματος. Οι στατικές μεταβλητές αρχικοποιούνται μόνο στην πρώτη κλήση της συνάρτησης.

```c
#include <stdio.h>

void foo() {

  static int counter; int i = 0; counter = 0;

  counter++; i++;

  printf("%d %d\n", counter, i);

}

int main() {

  foo(); foo(); foo();

  return 0;

}
```

Προσοχή: αν αρχικοποιήσουμε **εκτός** δήλωσης μεταβλητής τότε η τιμή δεν διατηρείται

Τι θα τυπώσει;

```
$ ./static
1 1
1 1
1 1
```

Κατηγορίες Μνήμης

Υπάρχουν 3 κατηγορίες μνήμης:

1. Η στοίβα (stack)
2. Ο σωρός (heap)
3. Η παγκόσμια / στατική μνήμη (global / static memory) - (σήμερα)

Η Παγκόσμια / Στατική Μνήμη (Global / Static Memory)

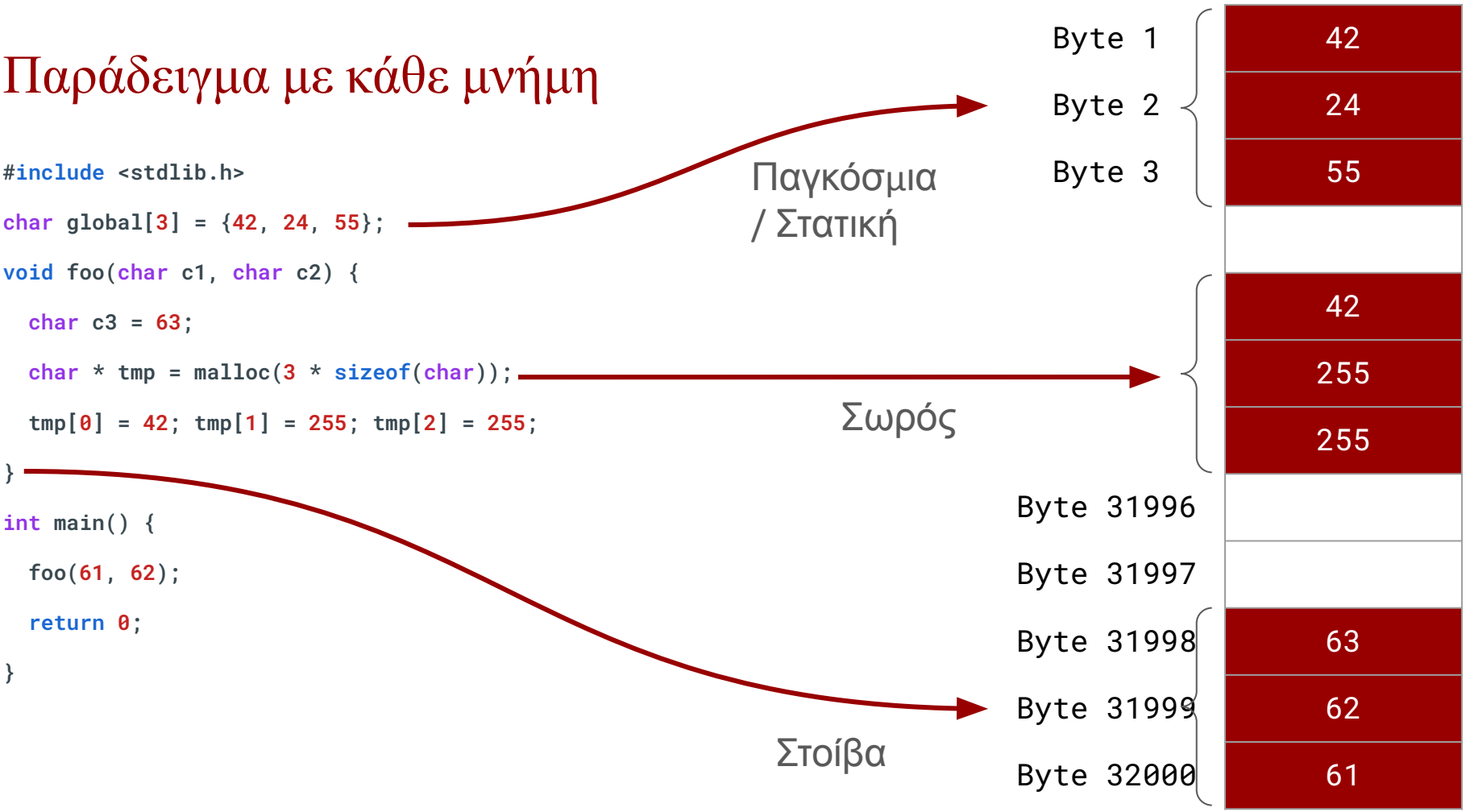
Η παγκόσμια / στατική μνήμη είναι ένα μέρος της μνήμης του προγράμματος ξεχωριστό από την στοίβα και στον σωρό που αποθηκεύονται παγκόσμιες, στατικές μεταβλητές καθώς και δεδομένα του προγράμματος
(ακόμα και ο κώδικας ο ίδιος).

Η μνήμη αυτή αρχικοποιείται όταν ξεκινά το πρόγραμμα και αποδεσμεύεται όταν τερματίσει.

Έχει πολλές υποκατηγορίες (data, code, .bss - άλλο μάθημα)

Παράδειγμα με κάθε μνήμη

```c
#include <stdlib.h>
char global[3] = {42, 24, 55};
void foo(char c1, char c2) {
  char c3 = 63;
  char * tmp = malloc(3 * sizeof(char));
  tmp[0] = 42; tmp[1] = 255; tmp[2] = 255;
}
int main() {
  foo(61, 62);
  return 0;
}
```



Τα περιεχόμενα της στατικής μνήμης

Κάθε αρχείο C μπορεί να ορίζει τις δικές του παγκόσμιες μεταβλητές και τα ονόματά τους σώζονται ως σύμβολα από τον μεταγλωττιστή.

```c
int global_buffer[10];

void foo() {

  static int counter = 0;

  counter++;

}

int main() {

  foo();

  return 0;

}
```

```
$ gcc -c memory.c
$ nm memory.o
0000000000000028 b counter.0
0000000000000000 T foo
0000000000000000 B global_buffer
0000000000000016 T main
$ du -sh memory.o
4.0K    memory.o
```

Τα περιεχόμενα της στατικής μνήμης

Κάθε αρχείο C μπορεί να ορίζει τις δικές του παγκόσμιες μεταβλητές και τα ονόματά τους σώζονται ως σύμβολα από τον μεταγλωττιστή.

```c
int global_buffer[100000000];

void foo() {

  static int counter = 0;

  counter++;

}

int main() {

  foo();

  return 0;

}
```

```
$ gcc -c memory.c
$ nm memory.o
0000000017d78400 b counter.0
0000000000000000 T foo
0000000000000000 B global_buffer
0000000000000016 T main
$ du -sh memory.o
4.0K    memory.o
```

Τα περιεχόμενα της στατικής μνήμης

Κάθε αρχείο C μπορεί να ορίζει τις δικές του παγκόσμιες μεταβλητές και τα ονόματά τους σώζονται ως σύμβολα από τον μεταγλωττιστή.

```c
int global_buffer[100000000] = {1};

void foo() {

  static int counter = 0;

  counter++;

}

int main() {

  foo();

  return 0;

}
```

Αν η μεταβλητή αρχικοποιηθεί με μη-μηδενικές τιμές, ο χώρος που χρειάζεται και όλες οι σχετικές τιμές αποθηκεύονται στο αρχείο από τον μεταγλωττιστή

```
$ gcc -c memory.c
$ nm memory.o
0000000000000000 b counter.0
0000000000000000 T foo
0000000000000000 D global_buffer
0000000000000016 T main
$ du -sh memory.o
382M    memory.o
```

Τα περιεχόμενα της στατικής μνήμης

Συμβολοσειρές που δεν χρησιμοποιούνται για την αρχικοποίηση ενός τοπικού πίνακα αποθηκεύονται επίσης στην στατική μνήμη.

```c
#include <stdio.h>

int main() {

  char * str1 = "Hello World", * str2 = "Hello World";

  printf("%p %p\n", str1, str2);

  return 0;

}
```

```
$ gcc -o const const.c
$ ./const
0x5588197ac004 0x5588197ac004
```

Τα περιεχόμενα της στατικής μνήμης

Συμβολοσειρές που δεν χρησιμοποιούνται για την αρχικοποίηση ενός τοπικού πίνακα αποθηκεύονται επίσης στην στατική μνήμη.

```c
#include <stdio.h>

int main() {

  char str1[] = "Hello World", str2[] = "Hello World";

  printf("%p %p\n", str1, str2);

  return 0;

}
```

```
$ gcc -o const const.c
$ ./const
0x7ffc2576ccc4 0x7ffc2576ccb8
```

Συναρτήσεις βιβλιοθήκης για συμβολοσειρές: string.h

Πίνακας Χαρακτήρων / Συμβολοσειρά (String)

Ένας πίνακας από χαρακτήρες λέγεται και αλφαριθμητικό / συμβολοσειρά (string). Λόγω της συχνής χρήσης τους, έχουμε αρκετές συντομεύσεις για αυτούς. Οι τρεις παρακάτω δηλώσεις είναι ισοδύναμες:

```
char hello[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\n', '\0'};
char hello[] = {72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 10, 0
char hello[] = "Hello World\n";
```

Προσοχή: τα string τερματίζονται πάντα με το **null byte**

'H'	'e'	'l'	'l'	'o'	' '	'W'	'o'	'r'	'l'	'd'	'\n'	'\0'

Η συνάρτηση strlen

Η συνάρτηση strlen επιστρέφει τον αριθμό των χαρακτήρων μιας συμβολοσειράς χωρίς να μετράει το null byte ('\0').

```
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  if (argc != 2) return 1;
  printf("Arg1 length: %d\n", strlen(argv[1]));
  return 0;
}
```

Η συνάρτηση strlen

Η συνάρτηση strlen επιστρέφει τον αριθμό των χαρακτήρων μιας συμβολοσειράς χωρίς να μετράει το null byte ('\0').

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  if (argc != 2) return 1;
  printf("Arg1 length: %d\n", strlen(argv[1]));
  return 0;
}
```

```
$ gcc -o strlen strlen.c
$ ./strlen "hello world"
```

Η συνάρτηση strlen

Η συνάρτηση strlen επιστρέφει τον αριθμό των χαρακτήρων μιας συμβολοσειράς χωρίς να μετράει το null byte ('\0').

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  if (argc != 2) return 1;
  printf("Arg1 length: %d\n", strlen(argv[1]));
  return 0;
}
```

```
$ gcc -o strlen strlen.c
$ ./strlen "hello world"
Arg1 length: 11
```

Η συνάρτηση strlen

Μια πιθανή υλοποίηση:

```
size_t strlen(char * str) {
  size_t length = 0;
  while(*str++) length++;
  return length;
}
```

Η συνάρτηση strcmp

Η συνάρτηση strcmp ελέγχει αν δύο συμβολοσειρές A και B είναι ίσες. Αν είναι επιστρέφει 0, αλλιώς επιστρέφει θετική τιμή αν το A > B (αρνητική αν B > A)

```c
#include <stdio.h>

#include <string.h>

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

  if (argc != 3) return 1;

  printf("strcmp(arg1, arg2): %d\n",

         strcmp(argv[1], argv[2]));

  return 0;

}
```

```
$ ./strcmp foo bar
strcmp(arg1, arg2): 4
$ ./strcmp foo foo
strcmp(arg1, arg2): 0
$ ./strcmp f bar
strcmp(arg1, arg2): 4
$ ./strcmp bar f
strcmp(arg1, arg2): -4
$ ./strcmp bar c
strcmp(arg1, arg2): -1
$ ./strcmp bar ba
strcmp(arg1, arg2): 114
```

Η συνάρτηση strcmp

Μια πιθανή υλοποίηση:

```
int strcmp(char * str1, char * str2) {
  while(*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
  }
  return *str1 - *str2;
}
```

Η συνάρτηση strcmp

Μια πιθανή υλοποίηση:

```
int strcmp(char * str1, char * str2) {
  while(*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
  }
  return *str1 - *str2;
}
```

Η συνάρτηση strcpy

Η συνάρτηση strcpy αντιγράφει (copy) μια συμβολοσειρά (2ο όρισμα) σε μια άλλη (1ο όρισμα).

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  char hello[16] = "Hello!";
  char world[16];
  strcpy(world, hello);
  printf("world: %s\n", world);
  return 0;
}
```

```
$ ./strcpy
world: Hello!
```

Η συνάρτηση strcpy

Η συνάρτηση strcpy αντιγράφει (copy) μια συμβολοσειρά (2ο όρισμα) σε μια άλλη (1ο όρισμα).

```
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  char hello[16] = "Hello!";
  char world[16];
  strcpy(world, hello);
  printf("world: %s\n", world);
  return 0;
}
```

```
$ ./strcpy
world: Hello!
```

Η συνάρτηση strcpy είναι **ιδιαίτερα επικίνδυνη** (ασφάλεια!) καθώς γράφει στην μνήμη χωρίς να ελέγχει αν υπάρχει επαρκής χώρος - πρέπει να ελέγξουμε εμείς. Ελαχιστοποιούμε την χρήση της (man strncpy για μια λίγο καλύτερη - αλλά πάλι κακή - εκδοχή).

Η συνάρτηση strcpy

Μια πιθανή υλοποίηση:

```
char * strcpy(char * dst, char * src) {
  char * original_dst = dst;
  while(*src) *dst++ = *src++;
  *dst = '\0';
  return original_dst;
}
```

Η συνάρτηση strcpy

Μια πιθανή υλοποίηση:

```
char * strcpy(char * dst, char * src) {
  char * original_dst = dst;
  while(*src) *dst++ = *src++;
  *dst = '\0';
  return original_dst;
}
```

Η συνάρτηση strcat

Η συνάρτηση strcat συνενώνει (concatenates) δύο συμβολοσειρές στην πρώτη ξεκινώντας την αντιγραφή του 2ου ορίσματος από το τέλος (null byte) της πρώτης συμβολοσειράς.

```
#include <stdio.h>

#include <string.h>

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

  char hello[16] = "Hello ";

  char world[16] = "World!";

  strcat(hello, world);

  printf("concat: %s\n", hello);

  return 0;

}
```

```
$ ./strcat
concat: Hello World!
```

Προσοχή: η strcat έχει ακριβώς τα ίδια προβλήματα με την strcpy. Ελέγχουμε πάντα ότι υπάρχει αρκετός χώρος στον πίνακα για την αντιγραφή.

Η συνάρτηση strcat

Μια πιθανή υλοποίηση:

```
char *strcat(char *dst, char *src) {
    char *original_dst = dst;
    while (*dst) dst++;
    while (*src) *dst++ = *src++;
    *dst = '\0';
    return original_dst;
}
```

Η συνάρτηση strcat

Μια πιθανή υλοποίηση:

```
char *strcat(char *dst, char *src) {
    char *original_dst = dst;
    while (*dst) dst++;
    while (*src) *dst++ = *src++;
    *dst = '\0';
    return original_dst;
}
```

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου καλύψαμε τις σελίδες 63-68, 93-96.

- Static variables and more
- Data segment and bss section (another time)
- Variable shadowing
- strcpy και strncpy
- strdup , strchr , strtok

Ευχαριστώ και καλή μέρα εύχομαι!
Keep Coding ;)