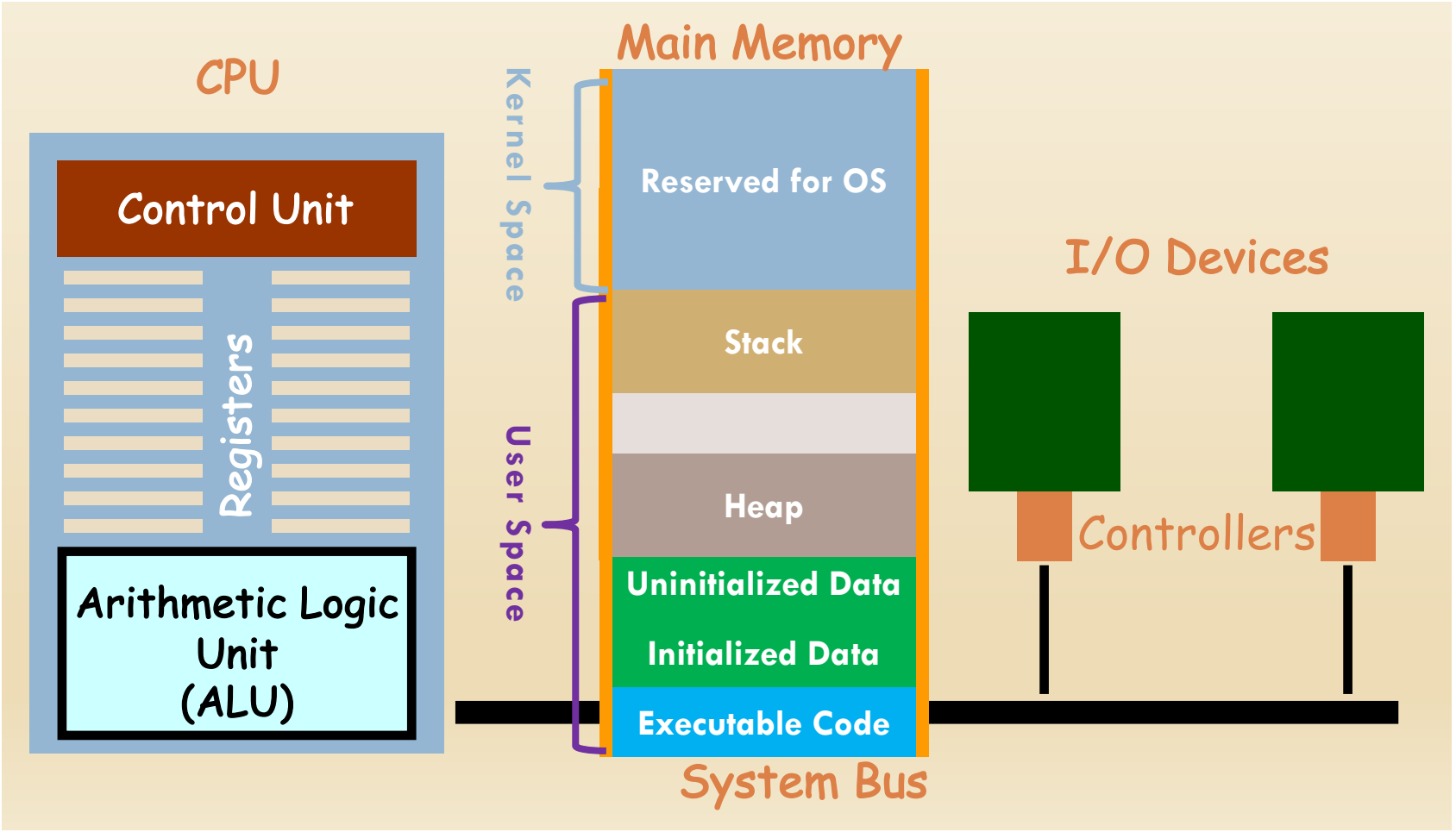
ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

13η Εβδομάδα: Ανασκόπηση Μαθήματος

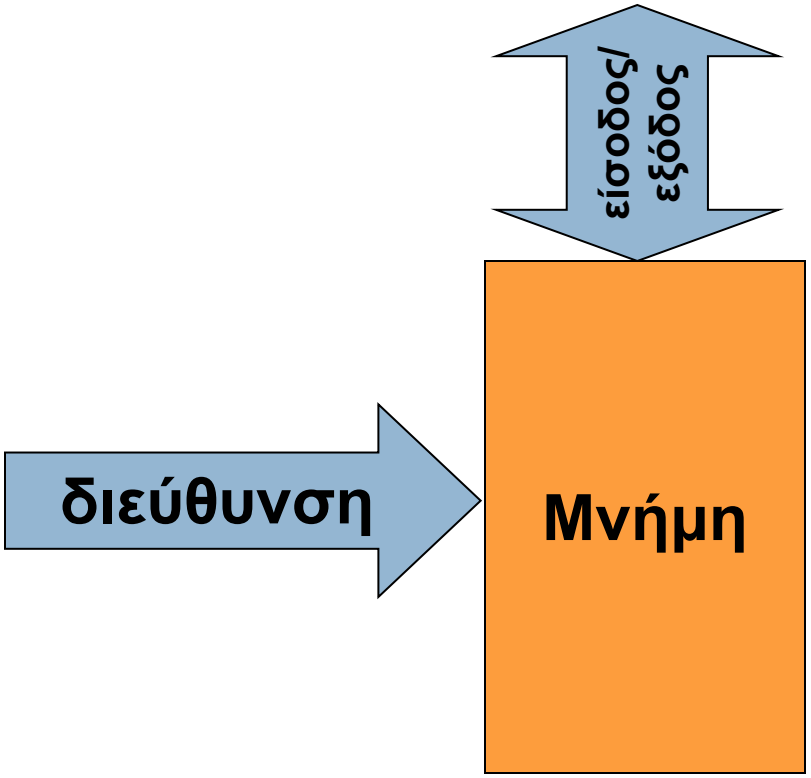
Αναφορές

Οι διαφάνειες της διάλεξης στηρίζονται, εν μέρει, σε υλικό παραδόσεων παλαιότερων ετών του **Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογία Υπολογιστών του Πανεπιστημίου Πατρών** καθώς και του **Τμήματος Πληροφορικής του Πανεπιστήμιο Κύπρου.**

Ο Υπολογιστής



Η Μνήμη

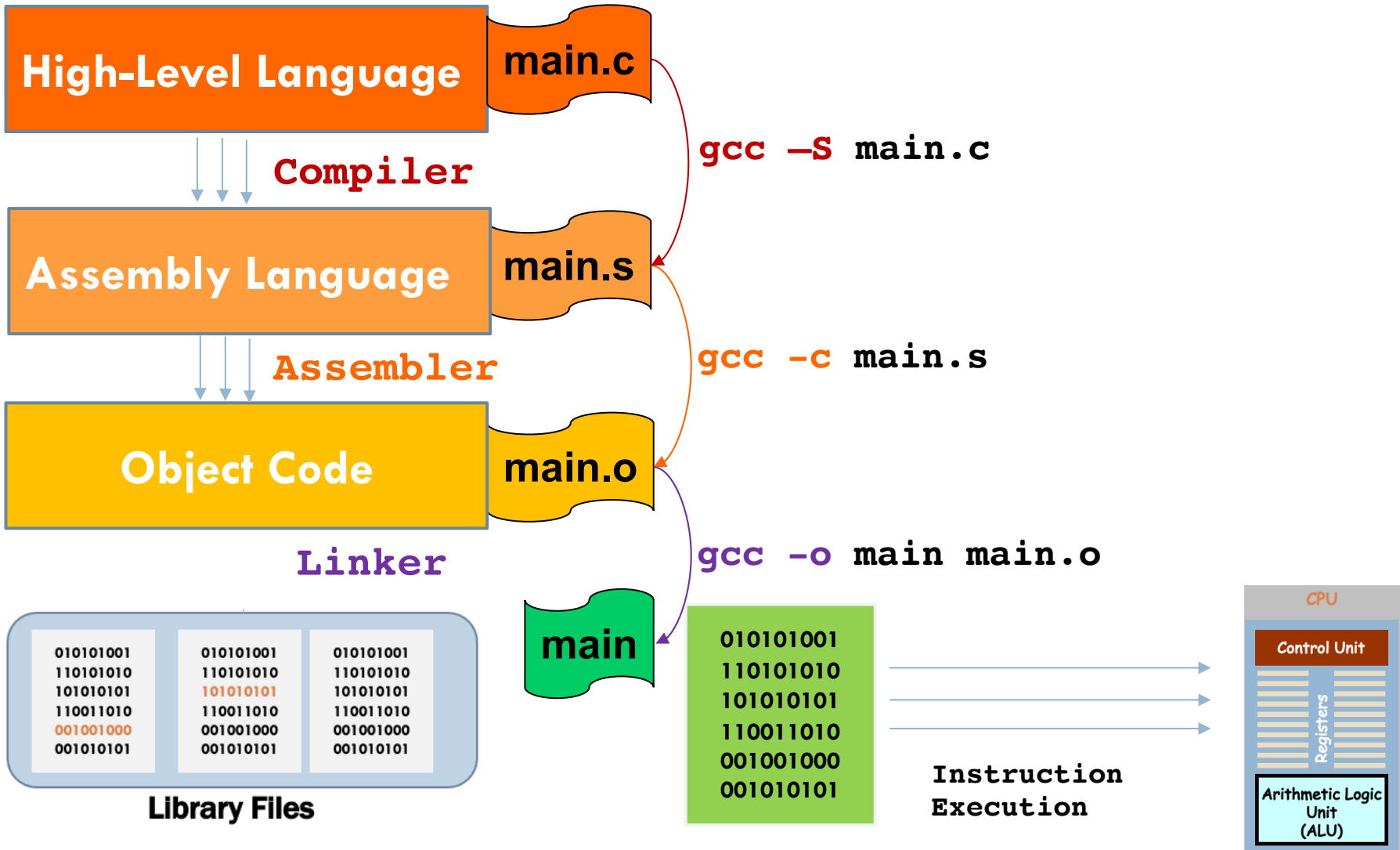


Διεύθυνση (address)	Περιεχόμενα
0x0000	00111111
0x0001	10110101
0x0002	11011110
0x0003	10111100
…	
0x00FF	00111101
0x0100	10101110
…	

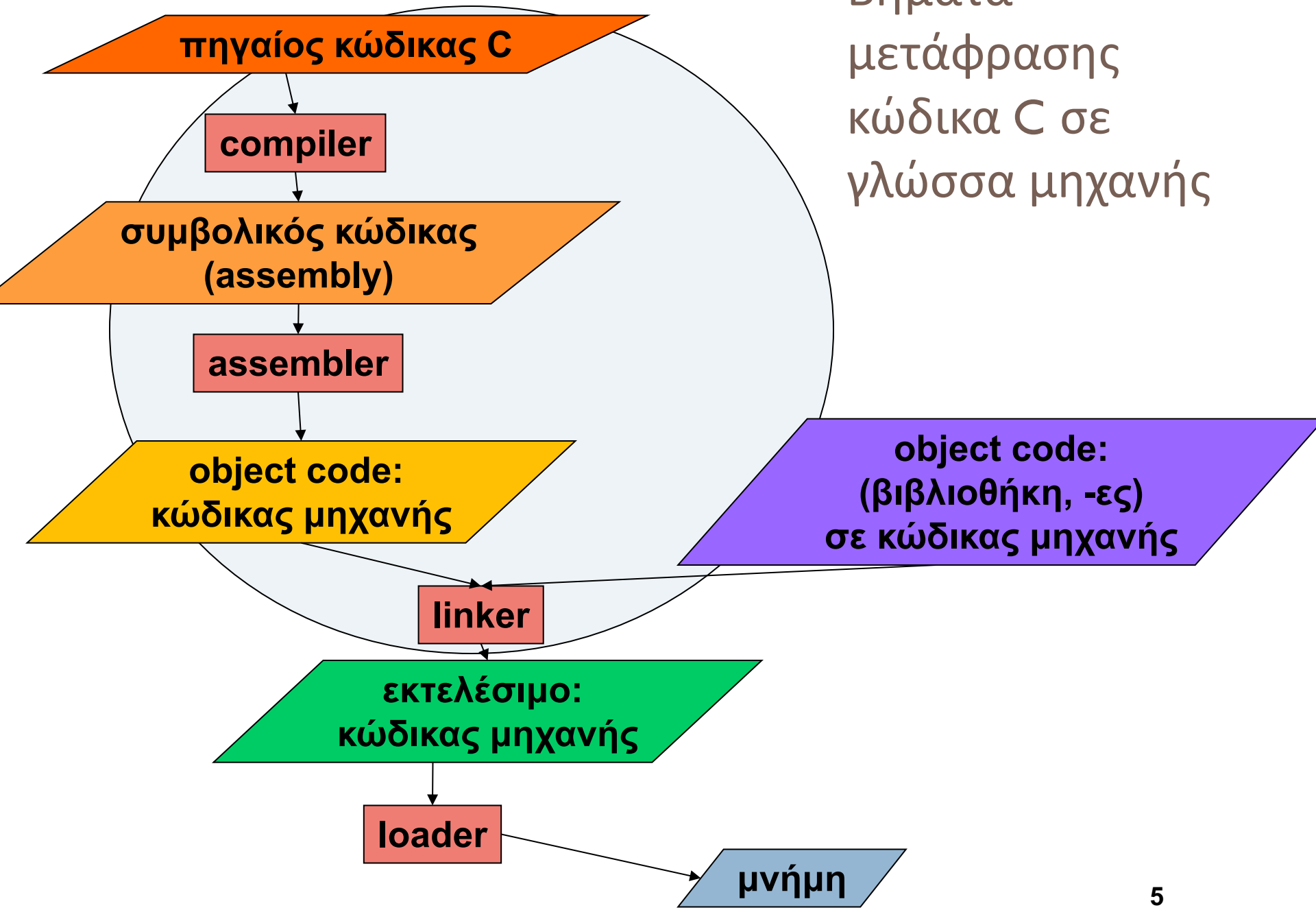
Πόσα bytes διαθέτονται για κάθε μεταβλητή;

Περιεχόμενα Μνήμης:
Δεδομένα και εντολές

Από γλώσσα υψηλού επιπέδου σε εκτελέσιμο



Βήματα μετάφρασης κώδικα C σε γλώσσα μηχανής



Παραδείγματα

main.c

```
#include <stdio.h>
#include <stdlib.h>
int main(){
   printf("Hello World");
   return 0;
}
```

main.o

- 010101001
- 110101010
- Call lib_func
- 101010101
- 110011010
- 001001000
- other lib call

main.s

```
·_main:                                 ## @main
·       .cfi_startproc
·## %bb.0:
·       pushq   %rbp
·       .cfi_def_cfa_offset 16
·       .cfi_offset %rbp, -16
·       movq    %rsp, %rbp
·       .cfi_def_cfa_register %rbp
·       xorl    %eax, %eax
·       movl    $0, -4(%rbp)
·       popq    %rbp
·       retq
·       .cfi_endproc
·                                       ## -- End function
```

Γλώσσα Assembly (Machine Specific)

1949

- **Data Movement Instructions**
  - mov, push, pop,...
- **Arithmetic and Logic Instructions**
  - add, sub,...
  - inc, dec,...
  - and, or, xor,...
- **Control Flow Instructions**
  - jmp,...
  - jcondition,...
  - cmp,...

Intel 8051 Microcontroller Instruction Set

DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

Πηγή:
https://www.electronicshub.org/8051-microcontroller-instruction-set/

Τύποι Δεδομένων

Ένας τύπος δεδομένων είναι ένα σύνολο τιμών και ένα σύνολο λειτουργιών (πράξεων) που μπορούν να εφαρμοστούν σε αυτές τις τιμές.

- Βασικοί τύποι δεδομένων
  - **char, int, float, double**
- Σύνθετοι τύποι δεδομένων: (πίνακες, δομές, ενώσεις)
  - **char [], int [], float [], double[],**
  - **struct x { int a;**

    float z;};

Δομές Εκτέλεσης

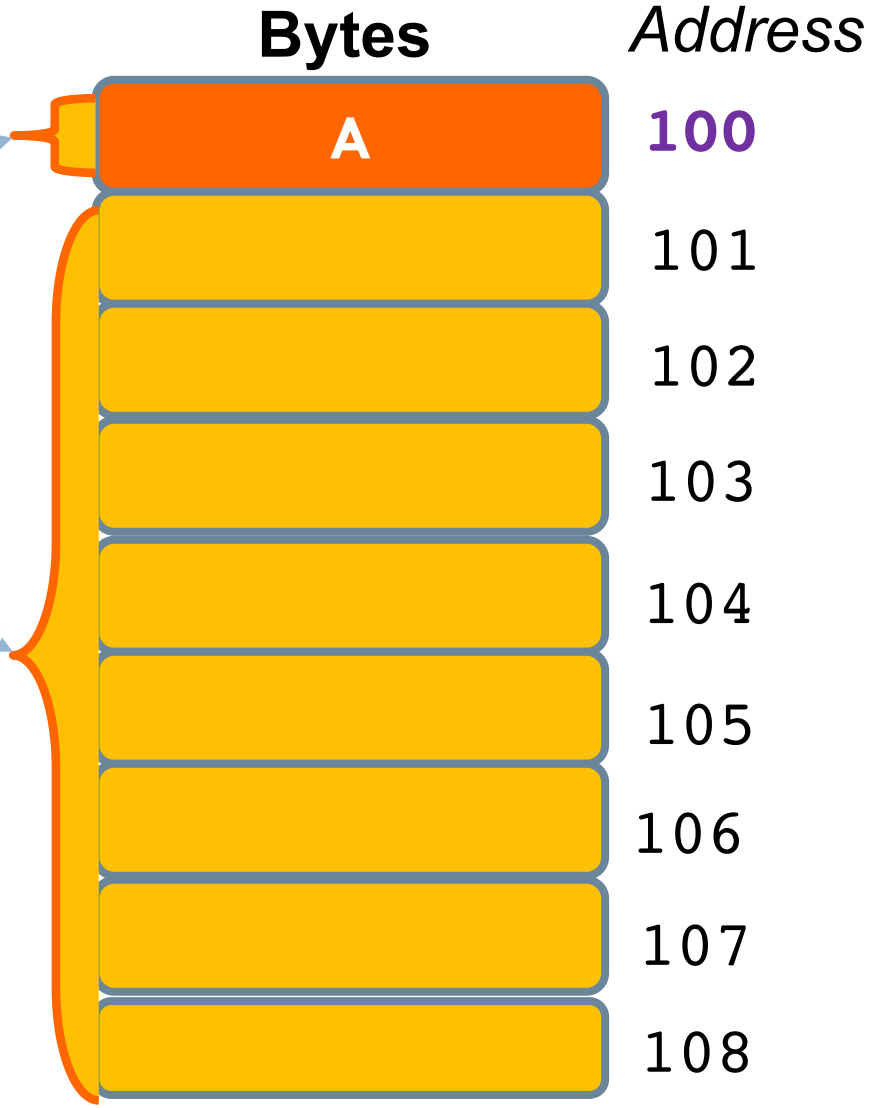
- **Διαδοχική εκτέλεση**
  - Στηρίζεται στην απλή παράθεση εκφράσεων/εντολών, η μια μετά την άλλη
- **Εκτέλεση με επιλογή**
  - Η ροή του προγράμματος “διακόπτεται” για να παρθεί μια απόφαση, να γίνει κάποια επιλογή
  - Το αποτέλεσμα της απόφασης καθορίζει την “κατεύθυνση” της ροής του προγράμματος
- **Εκτέλεση με επανάληψη**
  - Μια ομάδα εκφράσεων/εντολών εκτελείται περισσότερο από μια φορά

Τύποι Δεδομένων


```
char x;
x='A';
```

```
char *x_ptr;
```

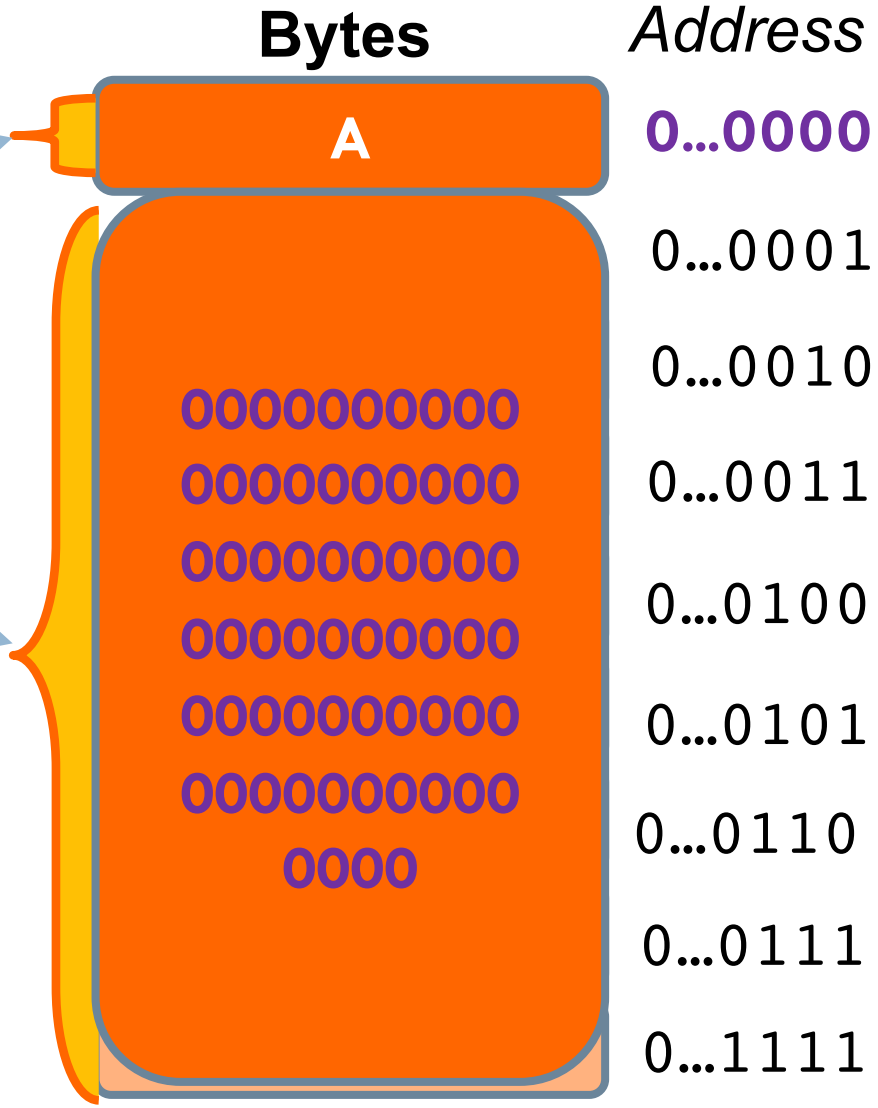
Γιατί ένας δείκτης δεσμεύει 8 bytes;



Τύποι Δεδομένων


```
char x;
x='A';
```

```
char *x_ptr;
x_ptr =&x;
```

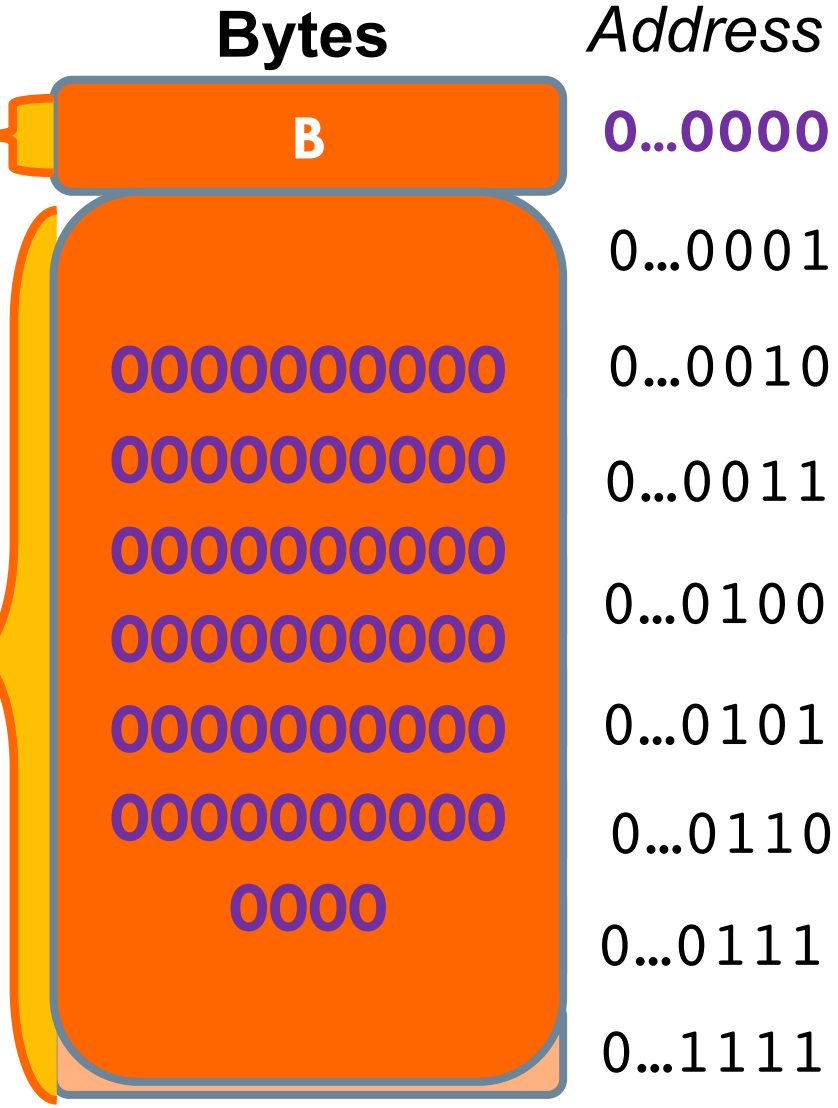


Τύποι Δεδομένων


```
char x;
x='A';
```

```
char *x_ptr;
x_ptr =&x;
```

```
* x_ptr ='B';
```



Πίνακες


```
char a[3]={'A','B','\0'};
```

a ⟺ &a[0]

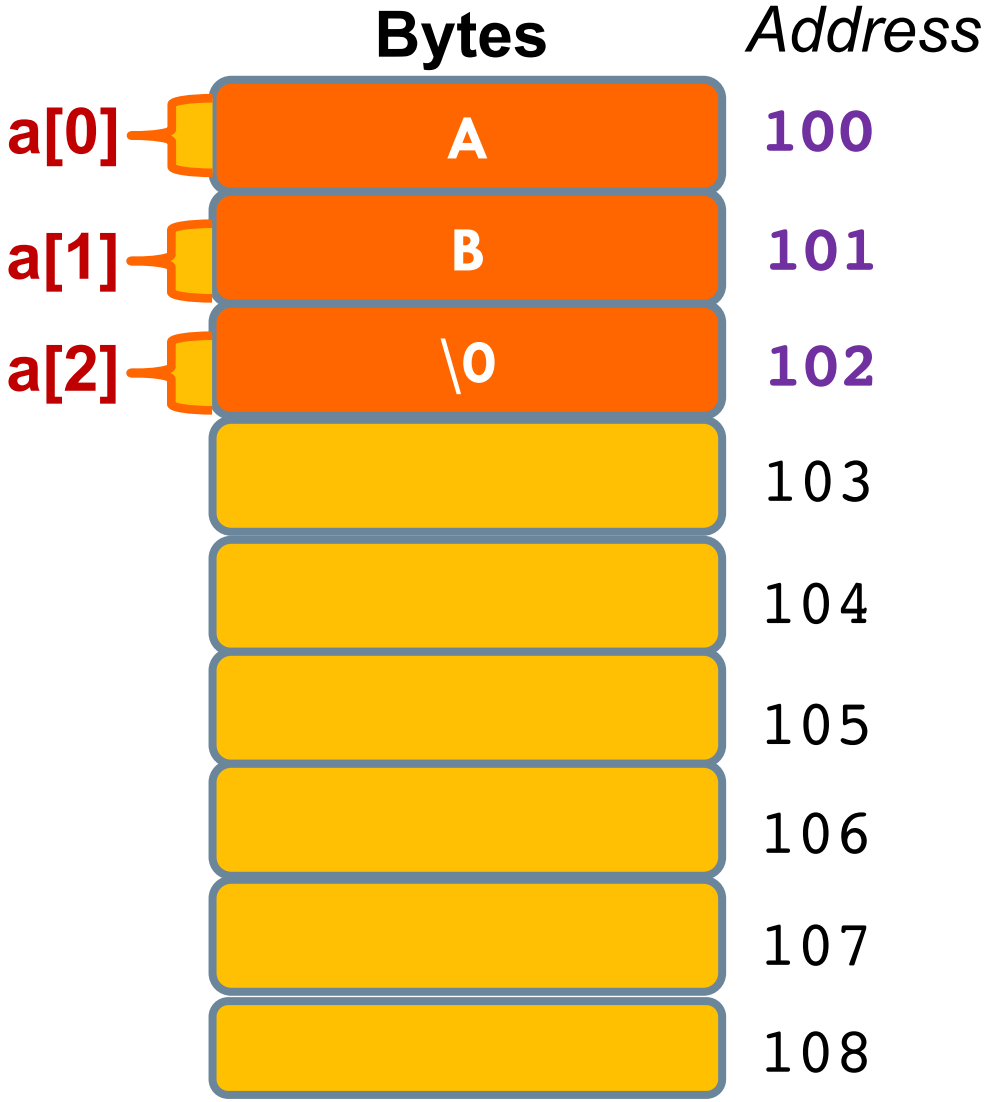
*a ⟺ a[0]

```
char *p;
p=&a[1];
```

p ⟺ &a[1]

*p ⟺ a[1]

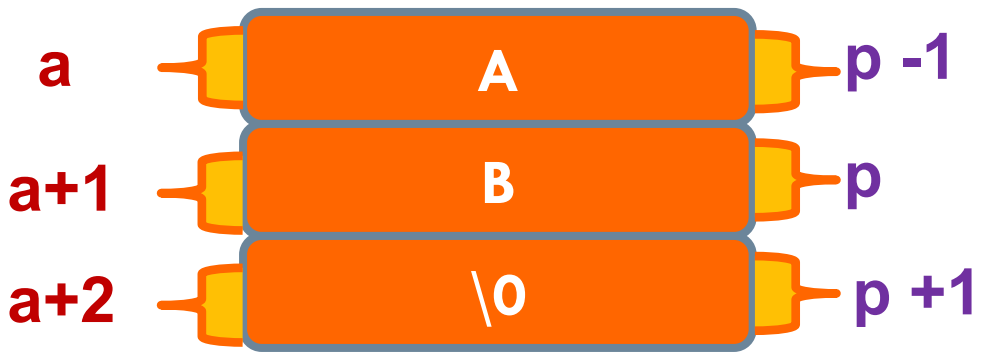
*(p-1) ⟺ a[0]



Πίνακες


```
char a[3]={'A','B','\0'};
char *p;
p=&a[1];
```

sizeof a: 3



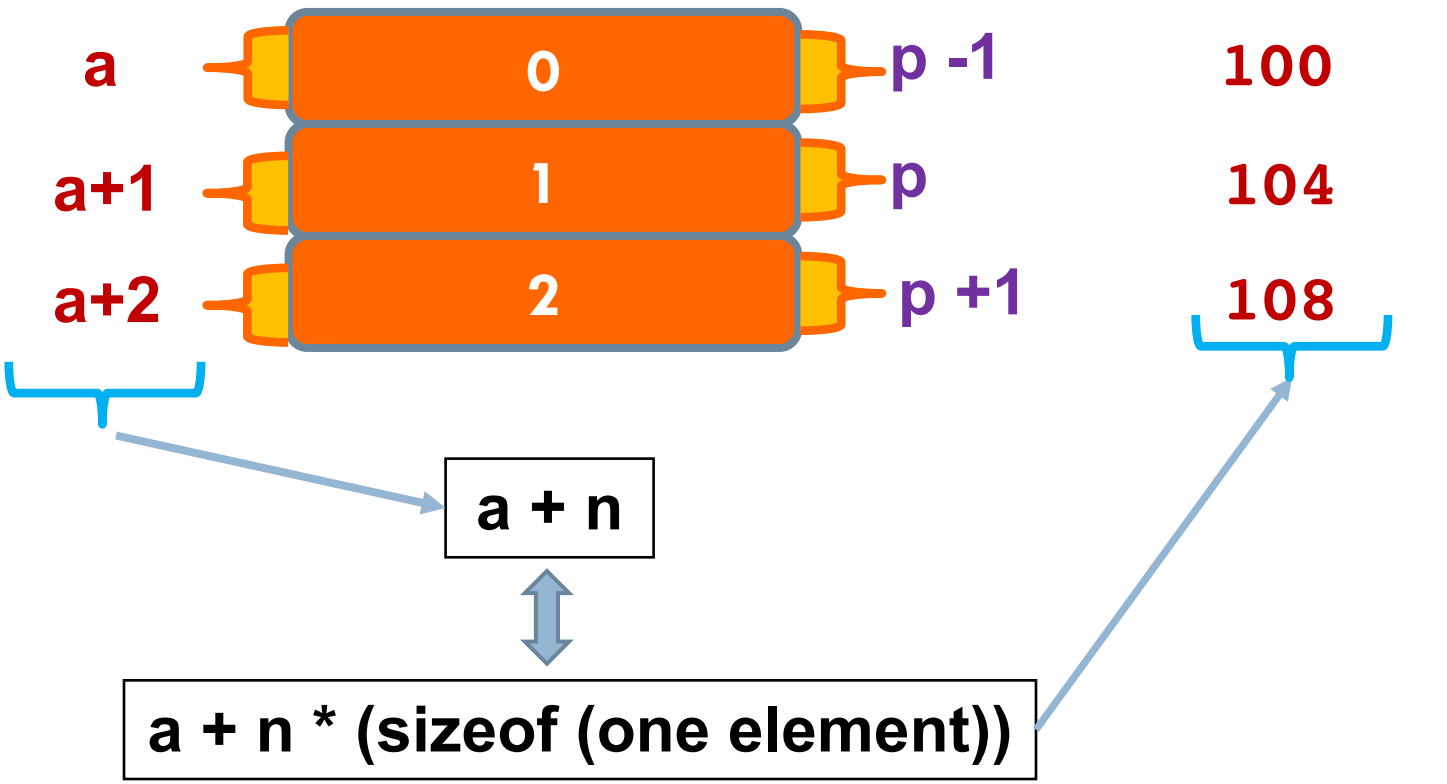
~~a=a+1;~~

Πίνακες


```
int a[3]={0,1,2};
```

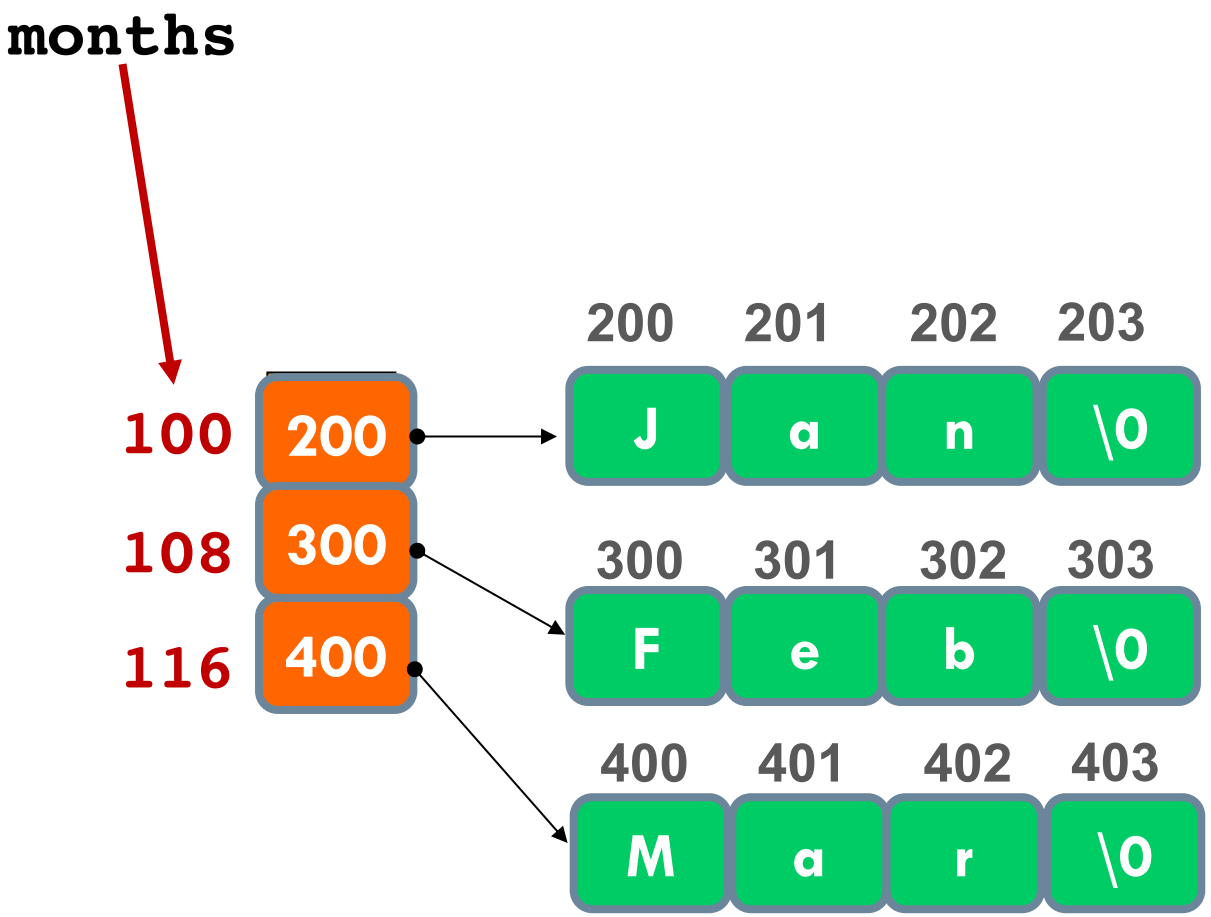
```
int *p;
p=&a[1];
```

sizeof a: 12



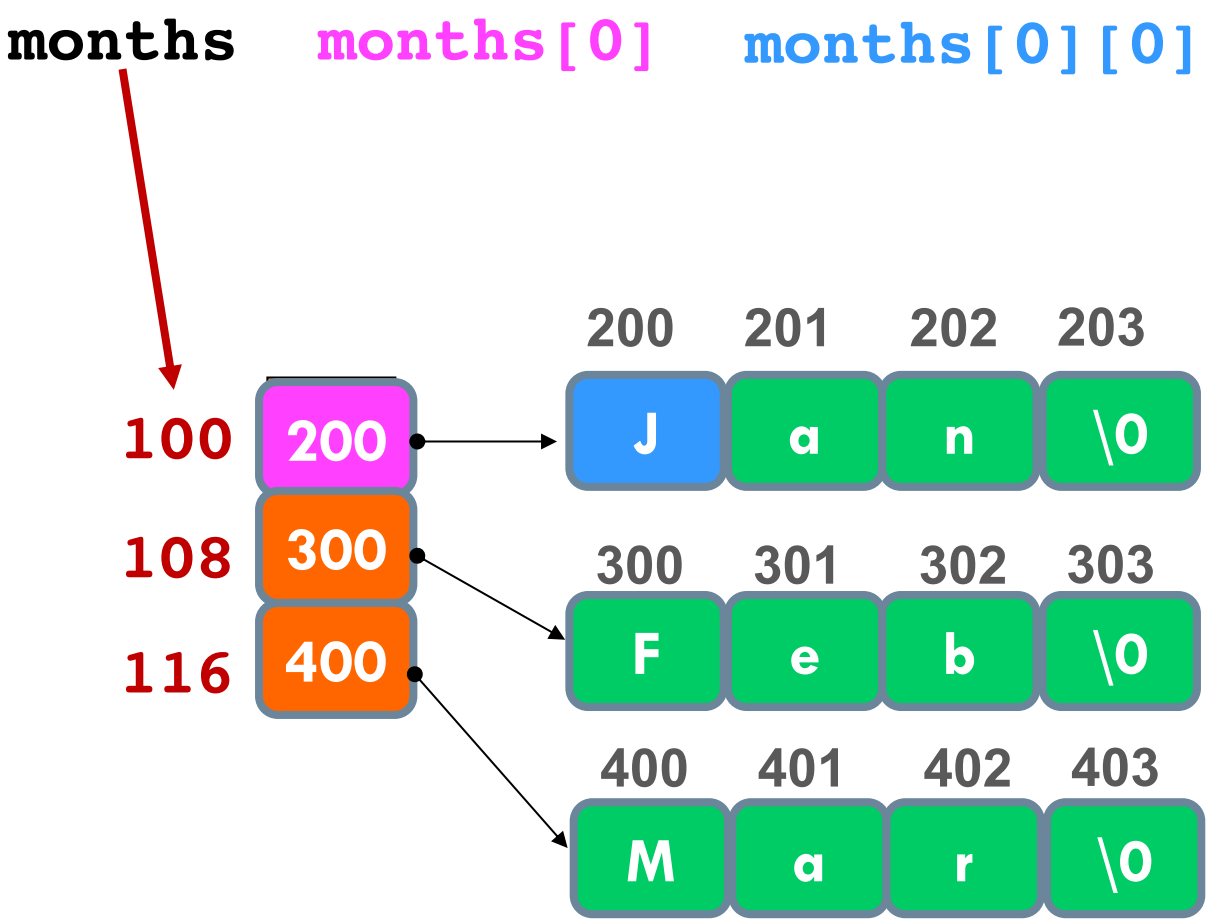
Πίνακας με δείκτες


```
char *months[]={“Jan”, “Feb”, “Mar”};
```



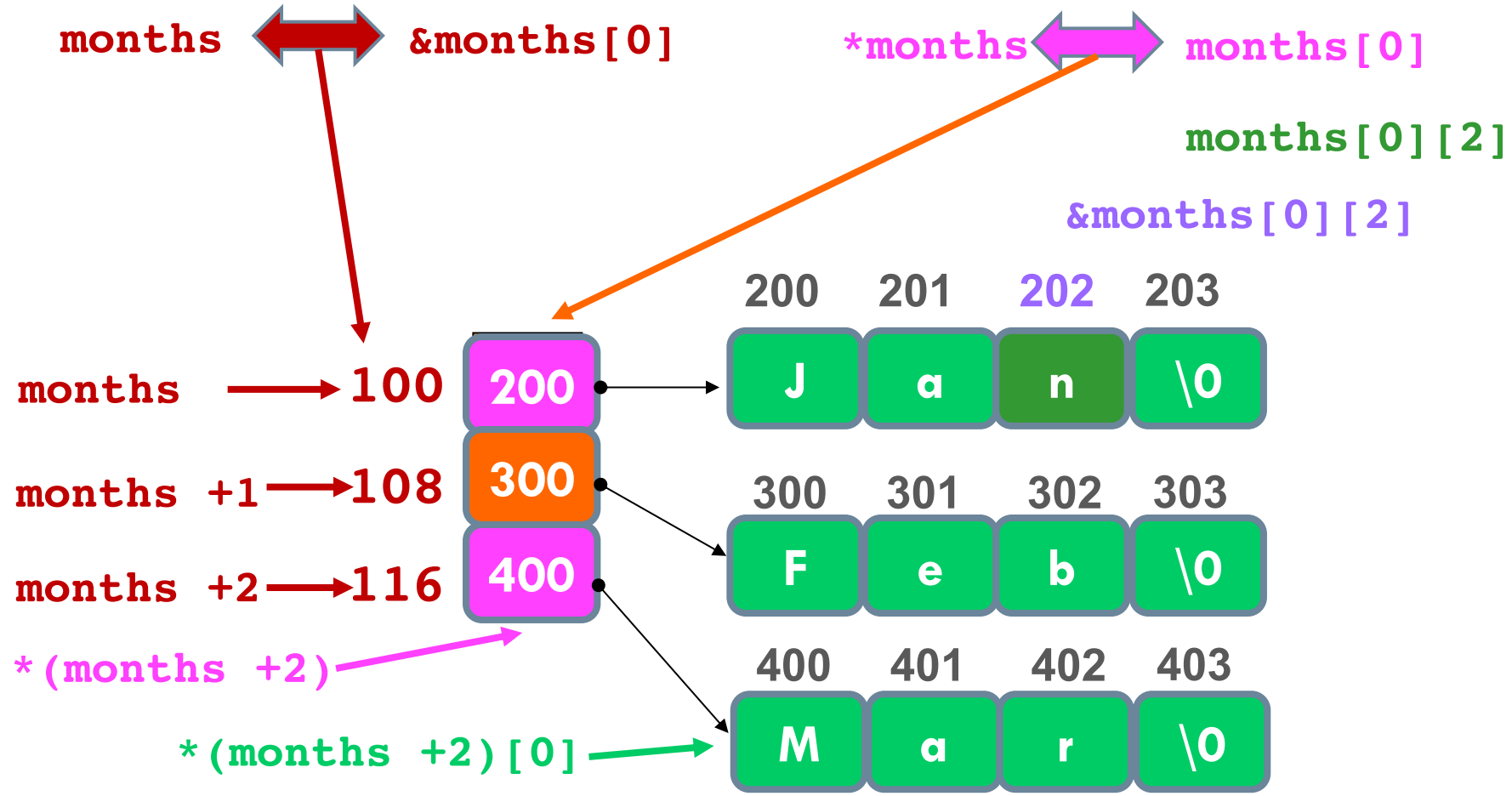
Πίνακας με δείκτες


```
char *months[]={“Jan”, “Feb”, “Mar”};
```



Πίνακας με δείκτες


```
char *months[]={"Jan", "Feb", "Mar"};
```



Δομές


```
typedef struct address {
 char street[5];
 int number;
 int code;
} Address;
```

```
Address x, *y;
```

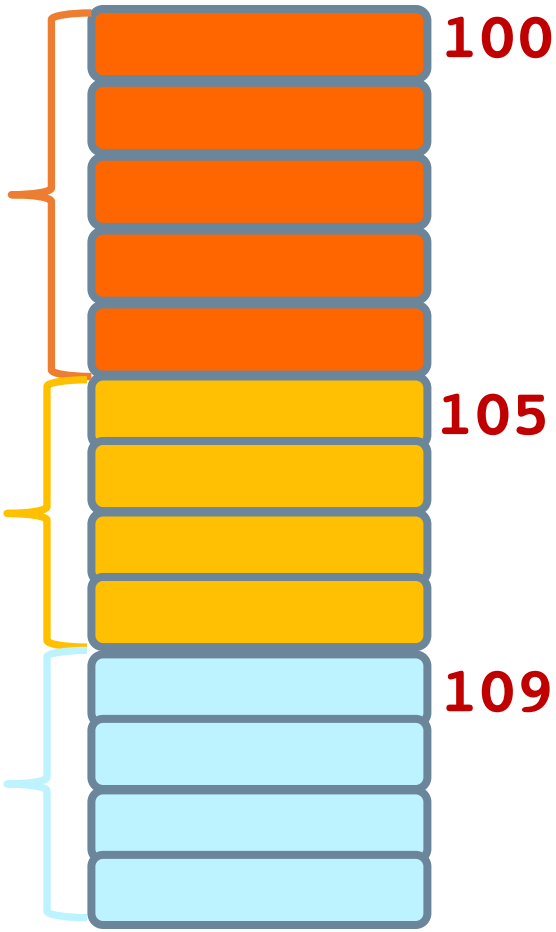
```
x.number=5;
```

```
x.code=7;
```

```
y=&x;
Y->code=6;
(*y).code=7;
```

(*p).μέλος_δομής = p->μέλος_δομής

sizeof (Address): 13



Παράδειγμα: *Σωστό ή Λάθος*

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct address {
 char street[20];
 int number;
 int code;
 char city[20];
} ;
typedef struct address Address;

void report (Address) ;
void readaddress (Address) ;

int main (void ) {

     Address myaddress ;

     readaddress(myaddress);
     report (myaddress);

return EXIT_SUCCESS;
}
```

```c
void readaddress (Address x)  {
     printf("Odos:\t");
     scanf("%s", x.street);
     printf("Ar. :\t");
     scanf("%d", & (x.number) );
     printf("Code:\t");
     scanf("%d", &(x.code) );
     printf("Poli:\t");
     scanf("%s", x.city);

     return;
}

void report (Address local) {
     printf("Odos: %20s\n", local.street);
     printf("Ar. : %20d\n", local.number);
     printf("T.C.: %20d\n", local.code);
     printf("Poli: %20s\n", local.city);
   return ;
}
```

Παράδειγμα: *Σωστό ή Λάθος*

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct address {
 char street[20];
 int number;
 int code;
 char city[20];
} ;
typedef struct address Address;

void report (Address) ;
void readaddress (Address *) ;

int main (void ) {

     Address myaddress ;

     readaddress(&myaddress);
     report (myaddress);

return EXIT_SUCCESS;
}
```

```c
void readaddress (Address *x)  {
     printf("Odos:\t");
     scanf("%s", x->street);
     printf("Ar. :\t");
     scanf("%d", &(x->number));
     printf("Code:\t");
     scanf("%d", &(x->code));
     printf("Poli:\t");
     scanf("%s", x->city);

     return;
}

void report (Address local) {
     printf("Odos: %20s\n", local.street);
     printf("Ar. : %20d\n", local.number);
     printf("T.C.: %20d\n", local.code);
     printf("Poli: %20s\n", local.city);
   return ;
}
```

Παράδειγμα: *Σωστό ή Λάθος*

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct address {
 char street[20];
 int number;
 int code;
 char city[20];
} ;
typedef struct address Address;

void report (Address) ;
void readaddress (Address *) ;

int main (void ) {

     Address myaddress ;

     readaddress(&myaddress);
     report (myaddress);

return EXIT_SUCCESS;
}
```

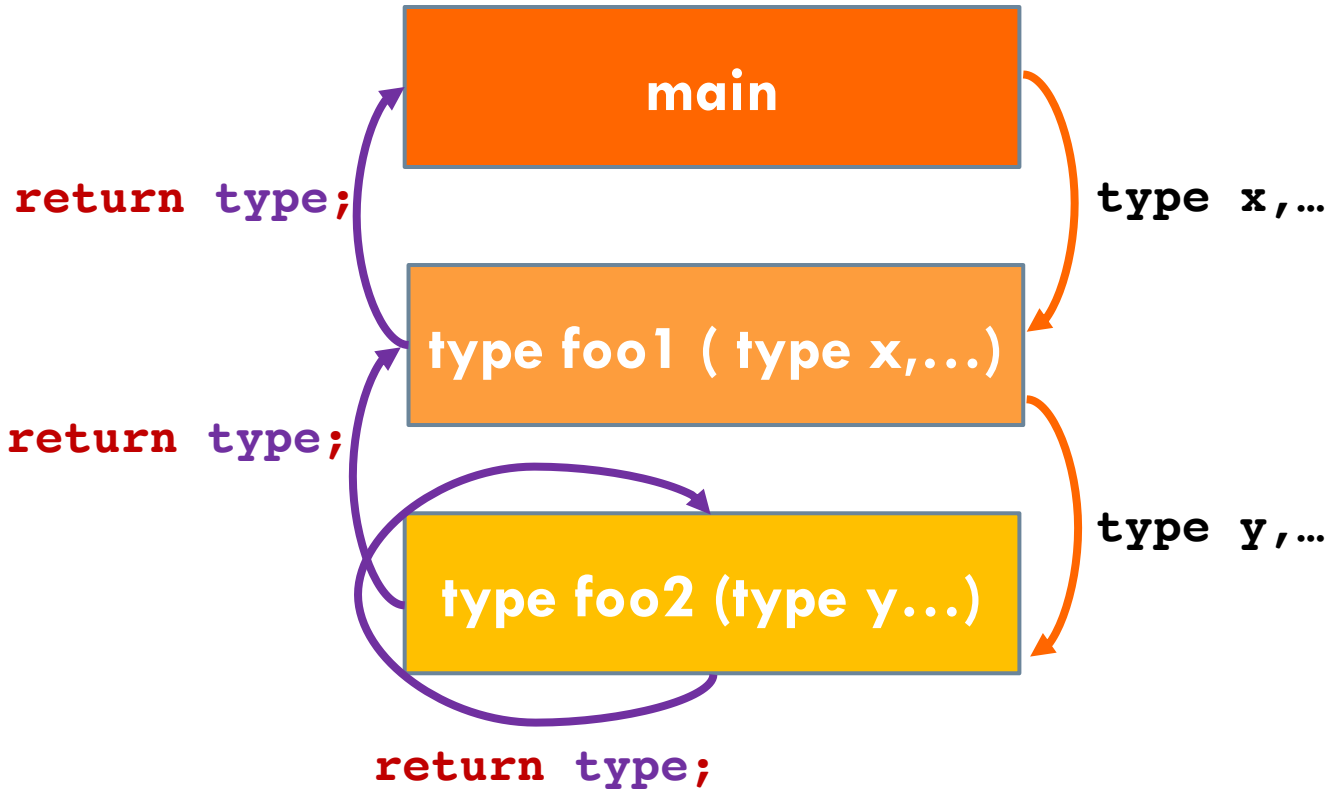
```c
void readaddress (Address *x)  {
     printf("Odos:\t");
     scanf("%s", (*x).street);
     printf("Ar. :\t");
     scanf("%d",  &((*x).number)) );
     printf("Code:\t");
     scanf("%d", &((*x).code) );
     printf("Poli:\t");
     scanf("%s", (*x).city);

     return;
}

void report (Address local) {
     printf("Odos: %20s\n", local.street);
     printf("Ar. : %20d\n", local.number);
     printf("T.C.: %20d\n", local.code);
     printf("Poli: %20s\n", local.city);
   return ;
}
```

Παράδειγμα: *Σωστό ή Λάθος*

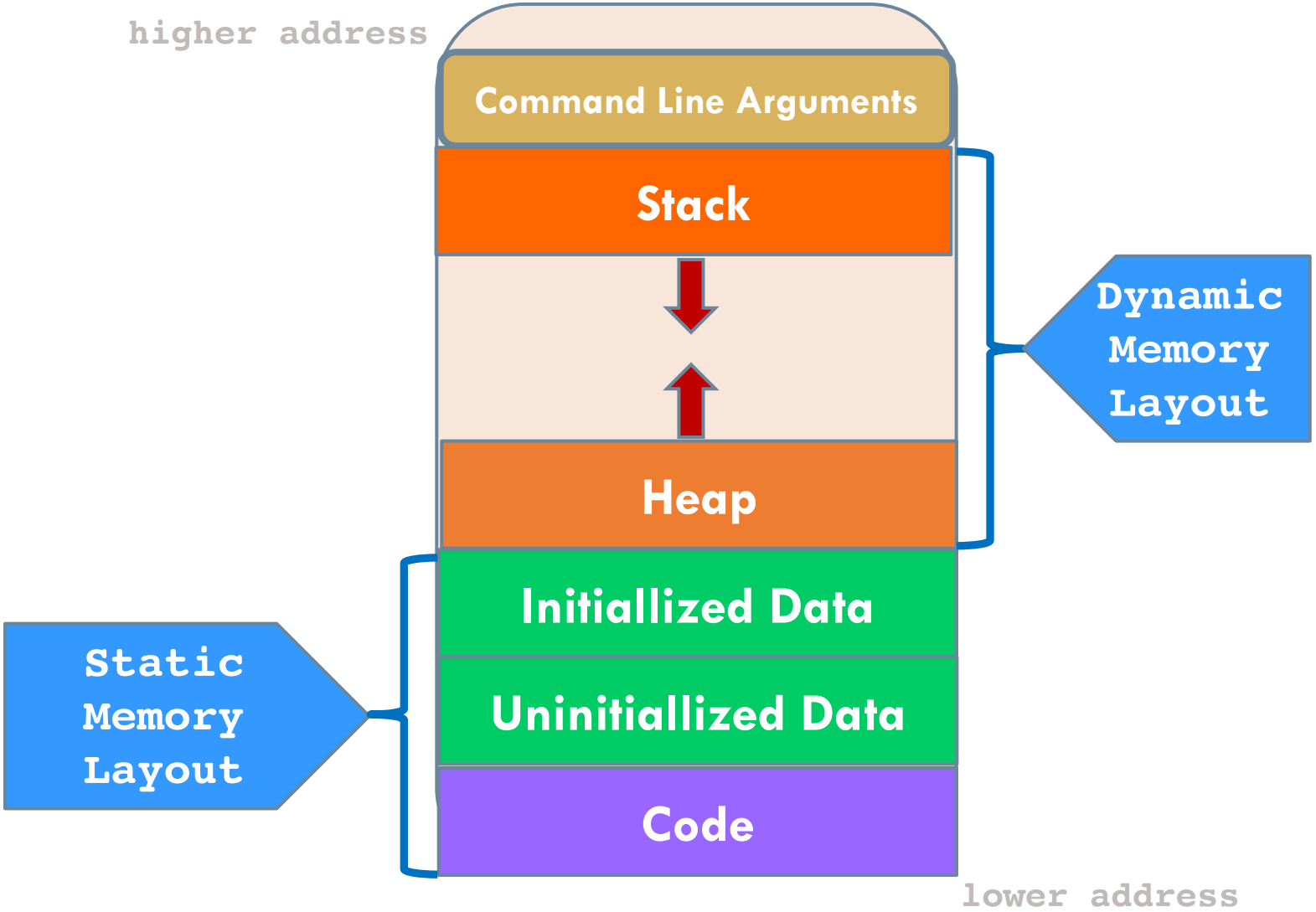
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct address {
 char street[20];
 int number;
 int code;
 char city[20];
} ;
typedef struct address Address;

void report (Address) ;
Address readaddress (void) ;

int main (void ) {

     Address myaddress ;

     myaddress = readaddress( );
     report (myaddress);

return EXIT_SUCCESS;
}
```

```c
Address readaddress (void)  {
     Address localaddress;
     printf("Odos:\t");
     scanf("%s", localaddress.street);
     printf("Ar. :\t");
     scanf("%d", &localaddress.number);
     printf("Code:\t");
     scanf("%d", &localaddress.code);
     printf("Poli:\t");
     scanf("%s", localaddress.city);

     return localaddress;
}

void report (Address local) {
     printf("Odos: %20s\n", local.street);
     printf("Ar. : %20d\n", local.number);
     printf("T.C.: %20d\n", local.code);
     printf("Poli: %20s\n", local.city);
   return ;
}
```

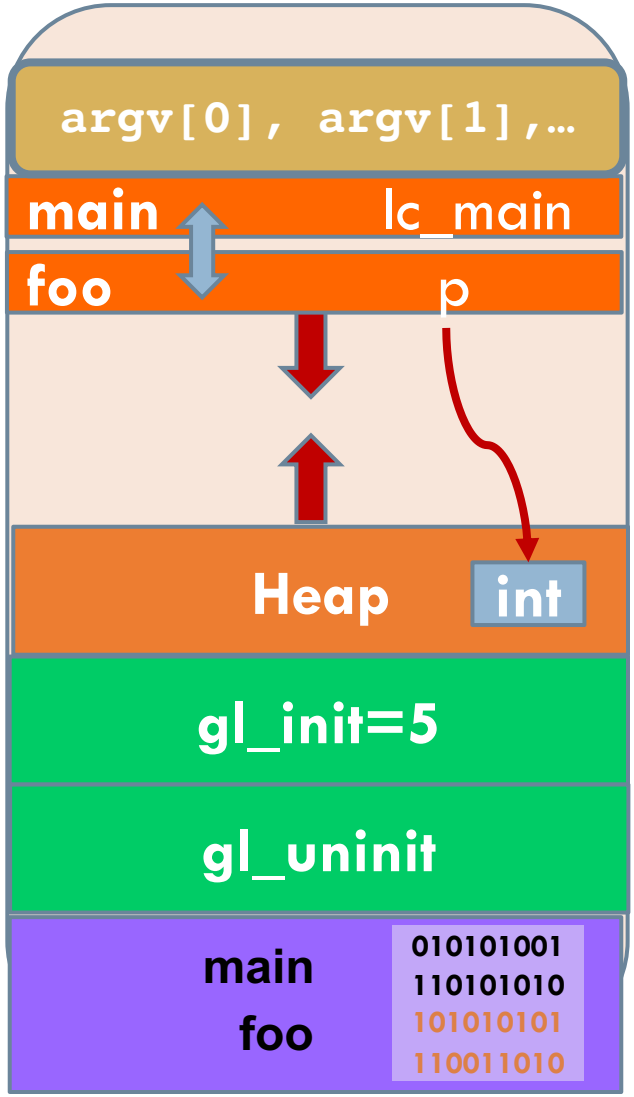
Διαδικαστικός Προγραμματισμός



Μνήμη και Πρόγραμμα

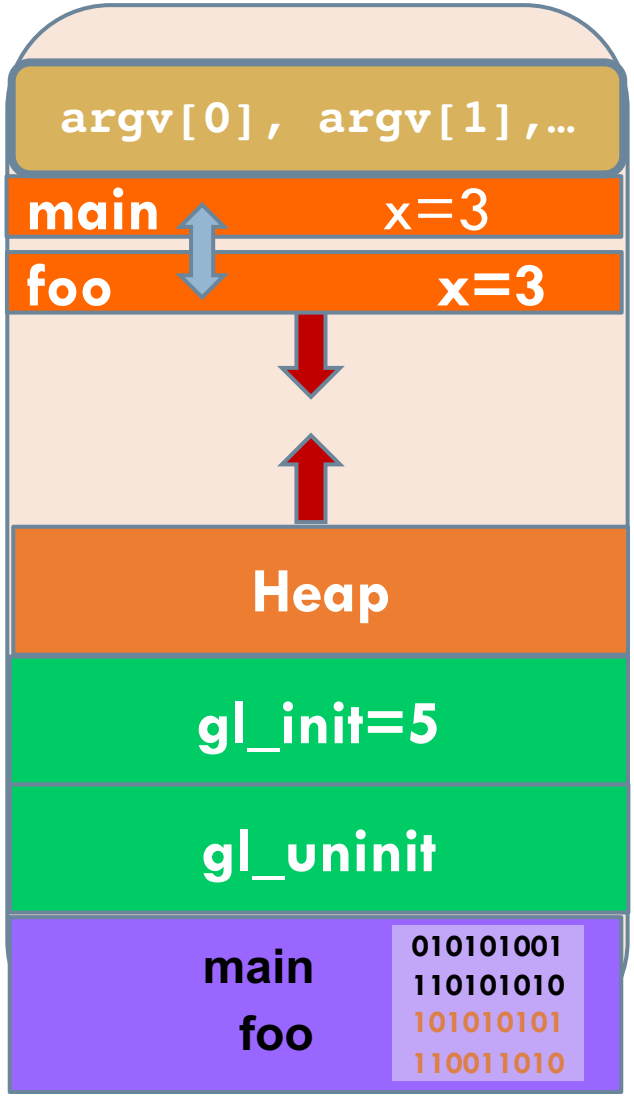


Μνήμη : Στοίβα και Σωρός



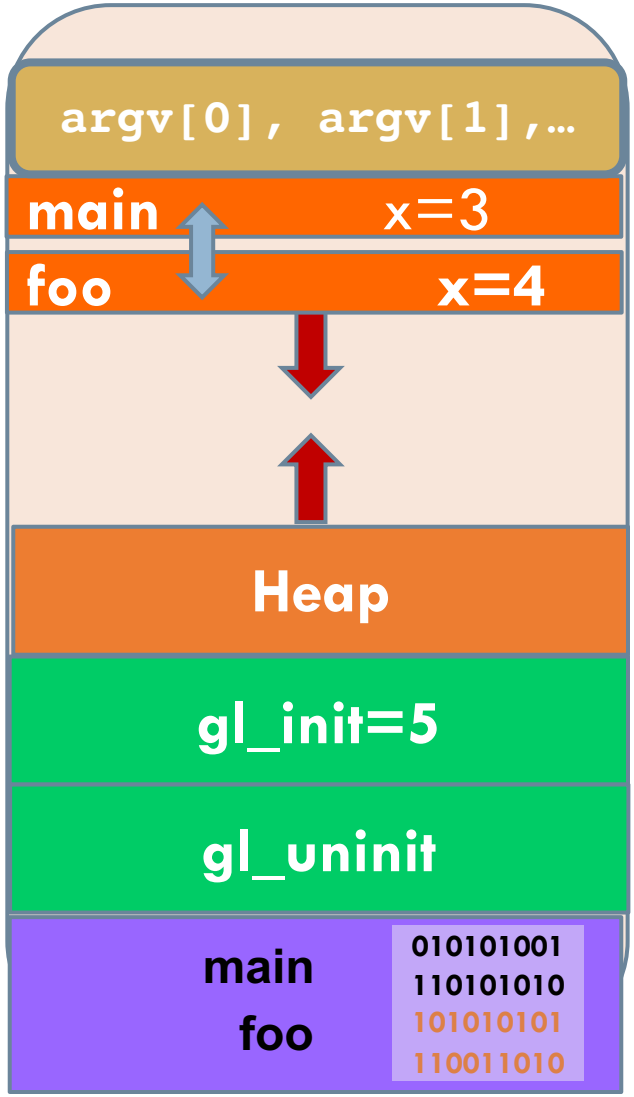

```
#include <stdio.h>
#include <stdlib.h>
int gl_init=5;
int gl_uninit;
int foo();
int main(int argc, char *argv[]){
    int lc_main;
    foo();
    return 0;
}

int foo(){
    int *p;
    p=(int *) malloc(sizeof(int));
}
```

Μνήμη: Κλήση Συνάρτησης με Τιμή



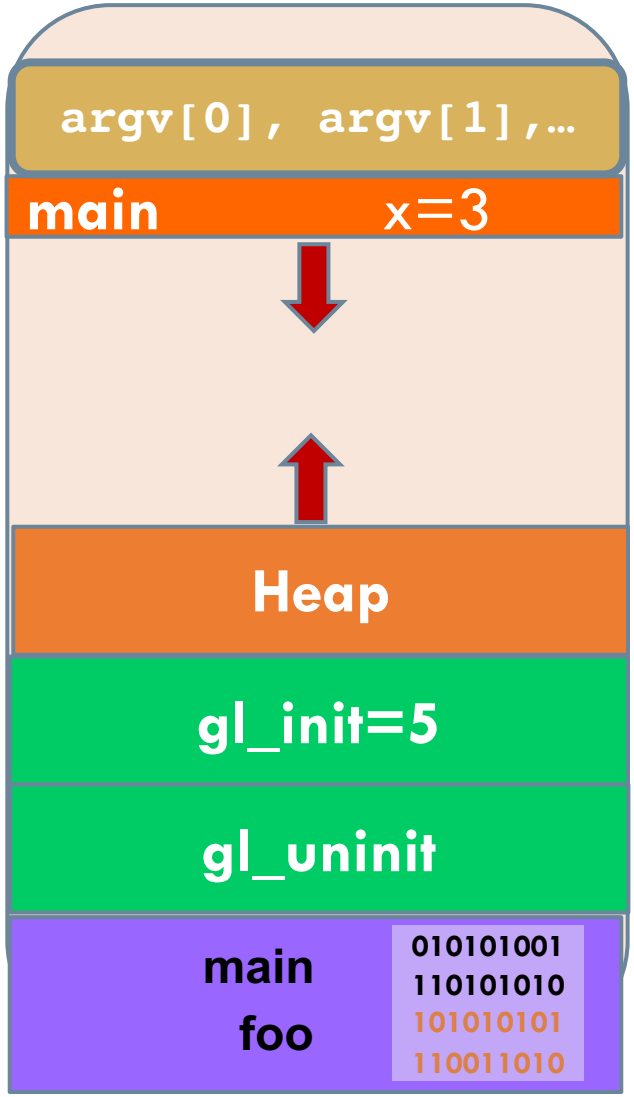

```
#include <stdio.h>
#include <stdlib.h>
int gl_init=5;
int gl_uninit;
void foo(int);
int main(int argc, char *argv[]){
    int x=3;
    foo(x);
    return 0;
}

void foo(int x){
    x=4;
    return;
}
```

Μνήμη: Κλήση Συνάρτησης με Τιμή



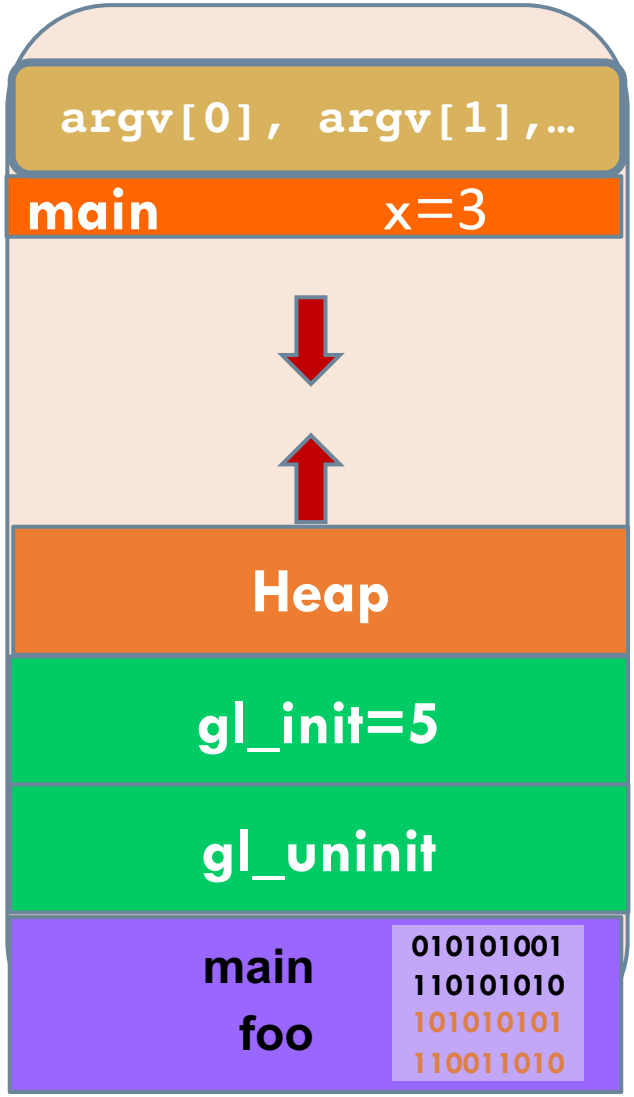

```
#include <stdio.h>
#include <stdlib.h>
int gl_init=5;
int gl_uninit;
void foo(int);
int main(int argc, char *argv[]){
    int x=3;
    foo(x);
    return 0;
}

void foo(int x){
    x=4;
    return;
}
```

Μνήμη: Κλήση Συνάρτησης με Τιμή



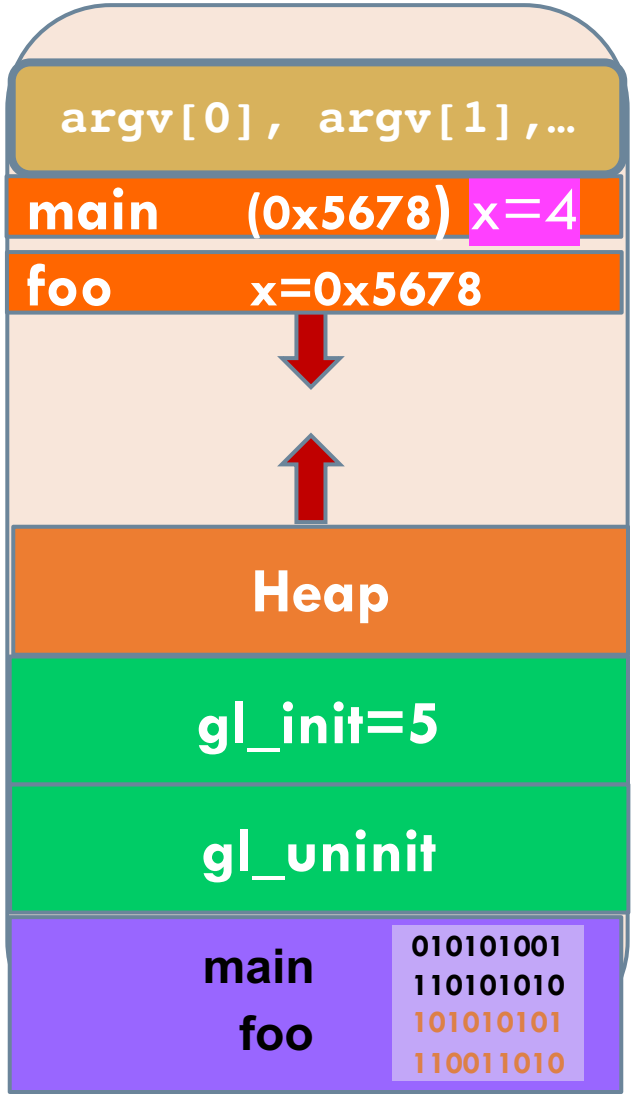

```
#include <stdio.h>
#include <stdlib.h>
int gl_init=5;
int gl_uninit;
void foo(int);
int main(int argc, char *argv[]){
    int x=3;
    foo(x);
    return 0;
}

void foo(int x){
    x=4;
    return;
}
```

Μνήμη: Κλήση Συνάρτησης με Αναφορά



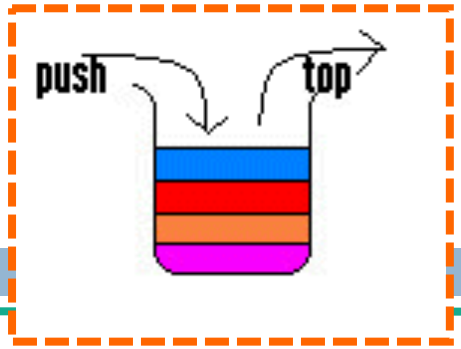

```
#include <stdio.h>
#include <stdlib.h>
int gl_init=5;
int gl_uninit;
void foo(int *);
int main(int argc, char *argv[]){
    int x=3;
    foo(&x);
    return 0;
}

void foo(int *x){
    *x=4;
    return;
}
```

Μνήμη: Κλήση Συνάρτησης με Αναφορά



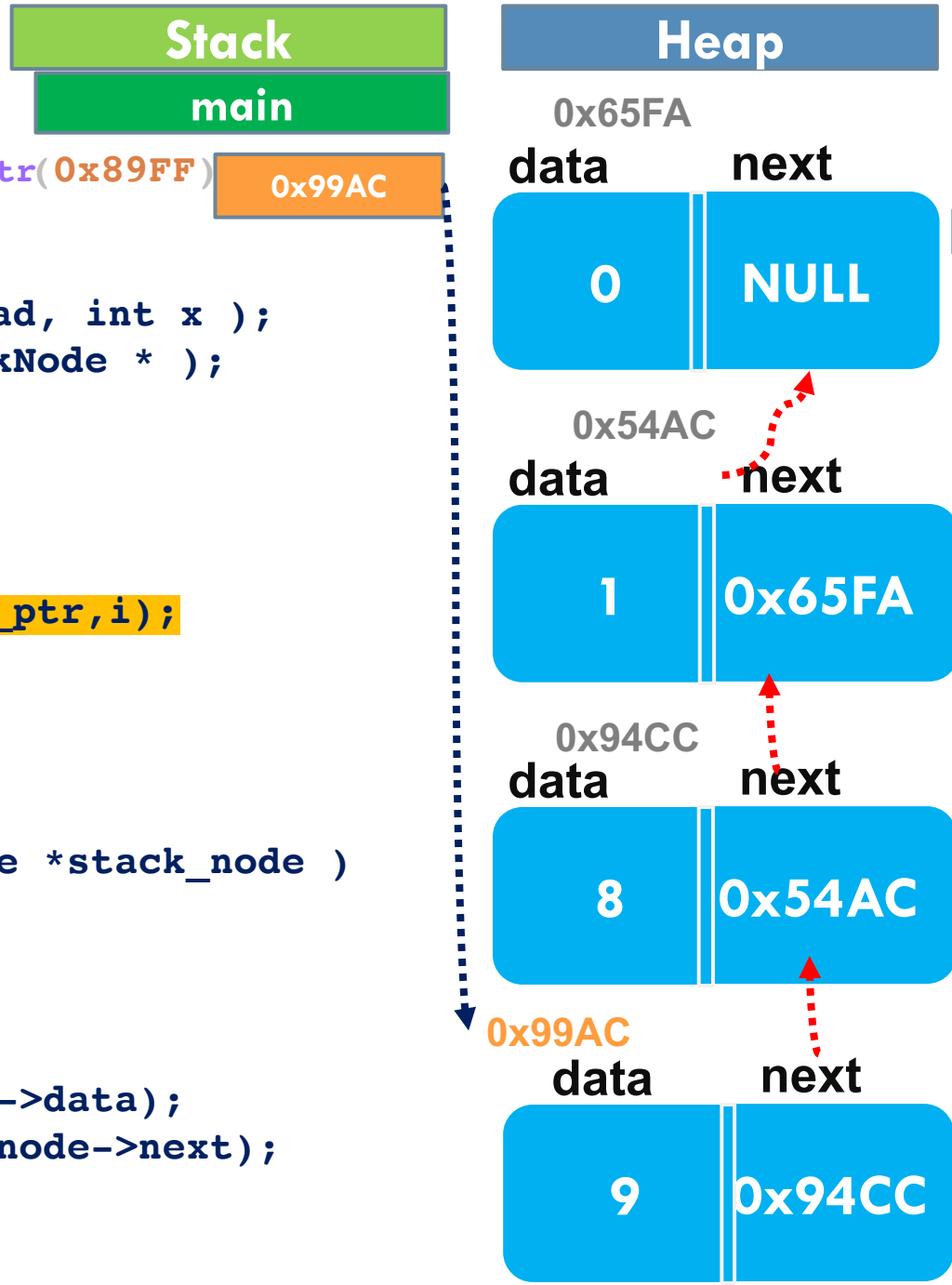

```
#include <stdio.h>
#include <stdlib.h>
int gl_init=5;
int gl_uninit;
void foo(int *);
int main(int argc, char *argv[]){
    int x=3;
    foo(&x);
    return 0;
}

void foo(int *x){
    *x=4;
    return;
}
```

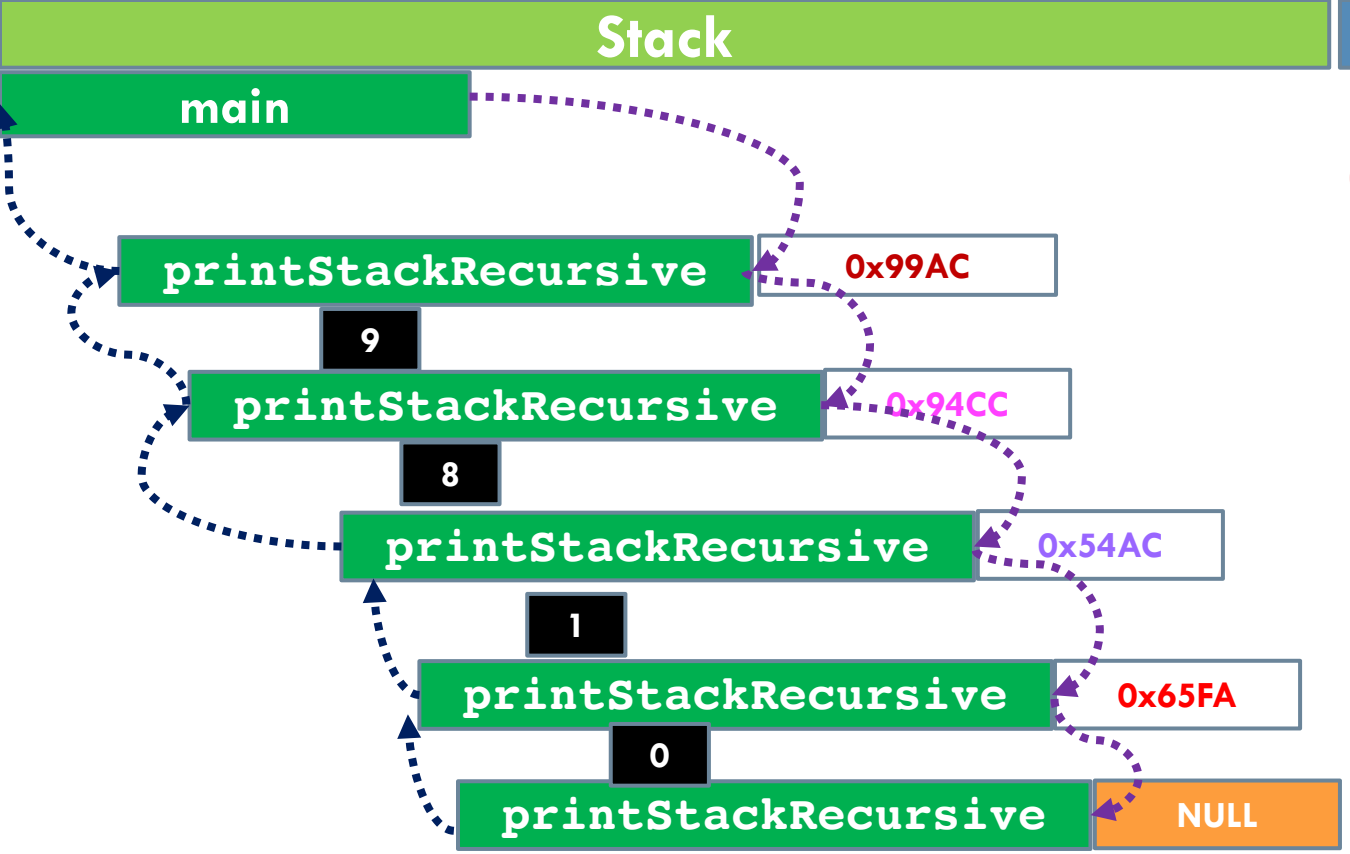
Εκτύπωση Στοίβας- Αναδρομή

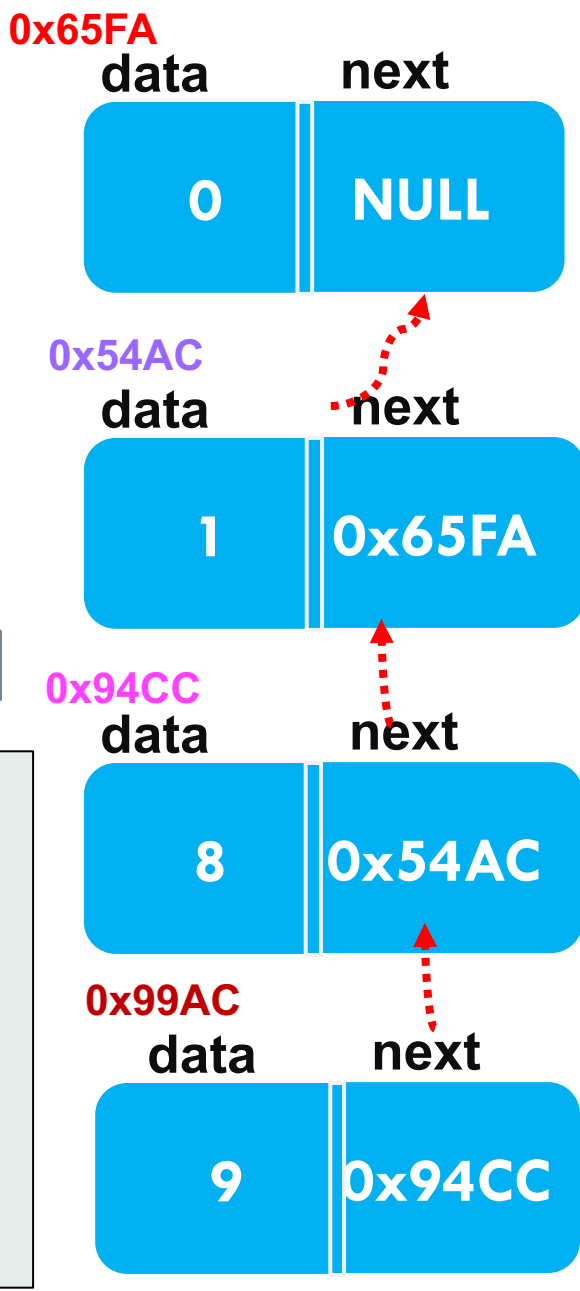



```
void printStackRecursive( StackNode *stack_node )
{
    if (stack_node == NULL )
        printf( "NULL\n\n" );
    else {
        printf("%d  -->", stack_node->data);
        printStackRecursive ( stack_node->next);
    }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;
void push(StackNode **stack_head, int x );
void printStackRecursive( StackNode * );

int main() {
    StackNode *top_ptr=NULL;
     int i;
    for(i=0;i<10;i++)push(&top_ptr,i);
printStackRecursive(top_ptr);
    return 0;
}

void printStackRecursive( StackNode *stack_node )
{
   if (stack_node == NULL )
      printf( "NULL\n\n" );
   else {
      printf("%d  -->", stack_node->data);
      printStackRecursive ( stack_node->next);
   } return;
}
```

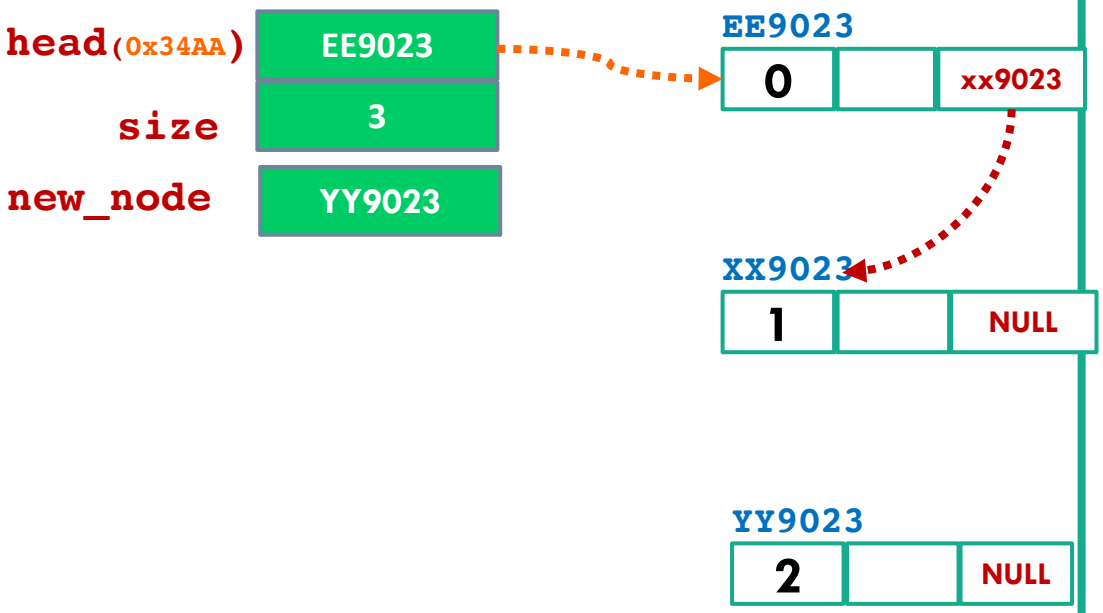







```
void printStackRecursive( StackNode *stack_node )
{
   if (stack_node == NULL )
      printf( "NULL\n\n" );
   else {
      printf("%d  -->", stack_node->data);
      printStackRecursive ( stack_node->nextPtr);
   }
   return;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
        int         val;
        struct node *next;
} NODE;

typedef struct {
     NODE  *head;
     int size;
} LIST;

void append_recursive(NODE **,NODE *);
int main()
{
    LIST mylist={NULL,0};
    NODE* new_node;
    int i;
    for(i=0;i<3;i++){
        new_node =createNode(&mylist, i);
        append_recursive(&(mylist.head), new_node);
     }
    return 0;
}
```

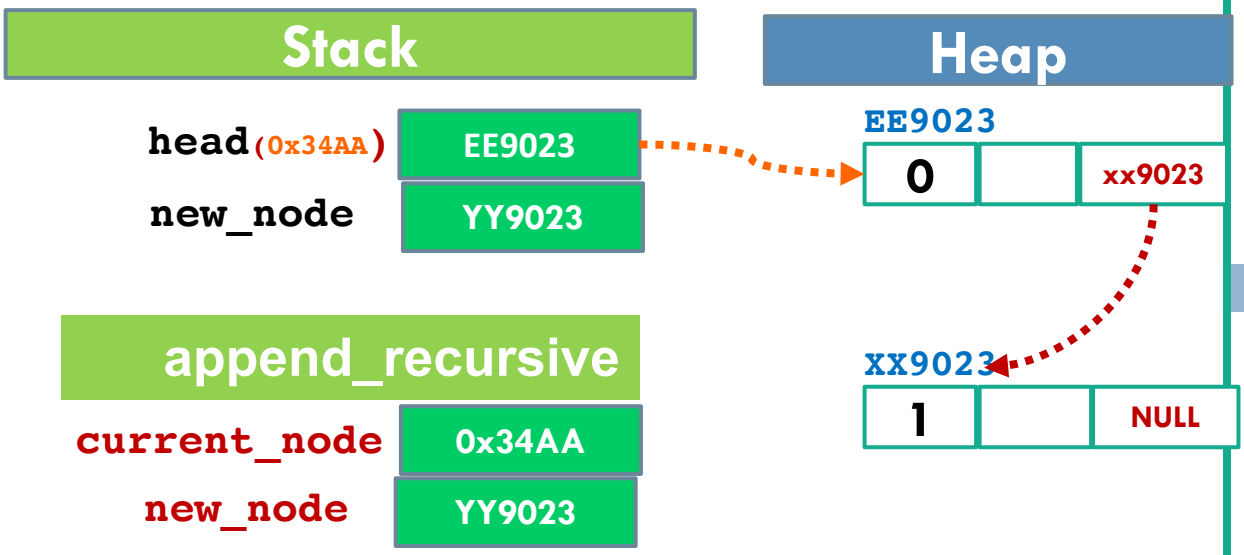


Προσθήκη Κόμβου με Αναδρομή στο τέλος Ευθύγραμμης Απλά Συνδεδεμένης Λίστας

Προσθήκη Κόμβου με Αναδρομή στο τέλος Ευθύγραμμης Απλά Συνδεδεμένης Λίστας

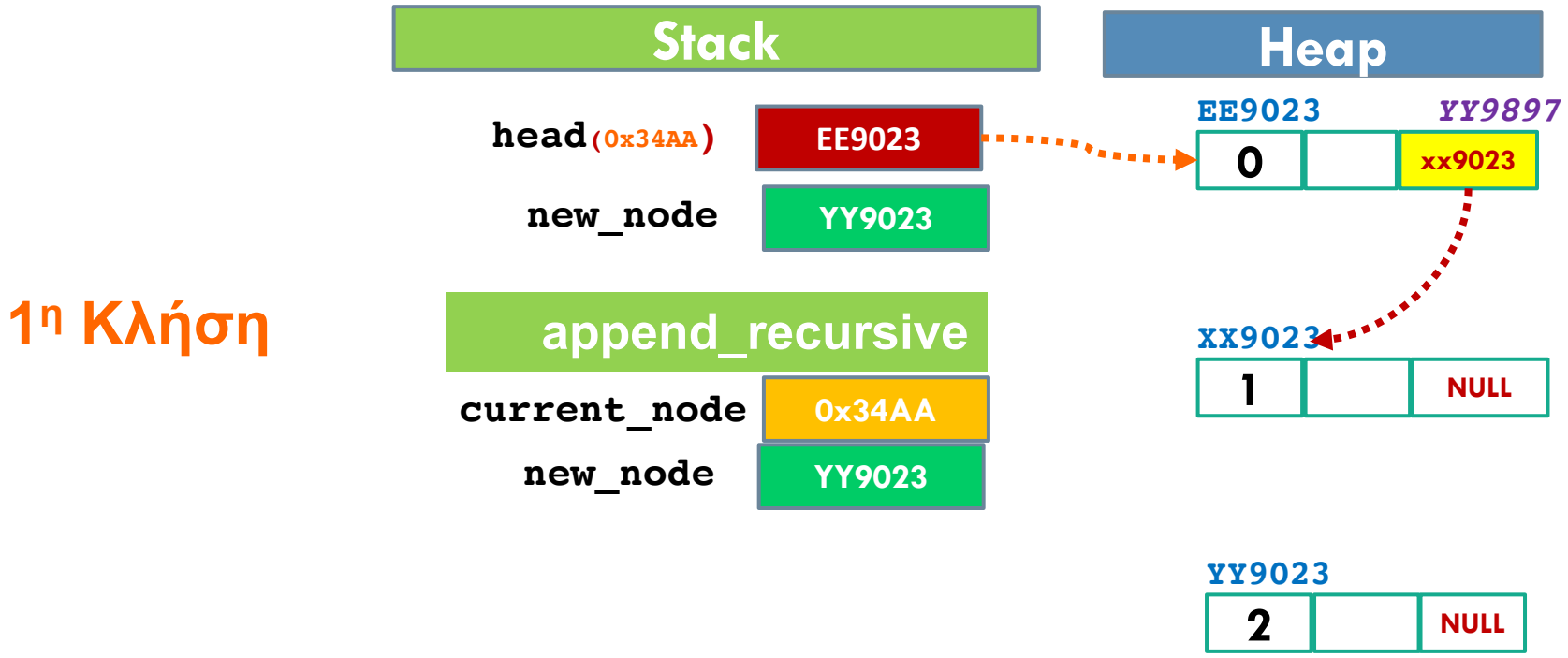

```
void append_recursive(NODE **current_node, NODE *new_node) {
            if (*current_node == NULL) {
                    *current_node = new_node;
             return ;
            }
            append_recursive( & ( (*current_node)->next ),
new_node );
          return;
}
```

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

typedef struct node {
    int     val;
    struct node *next;
} NODE;
```



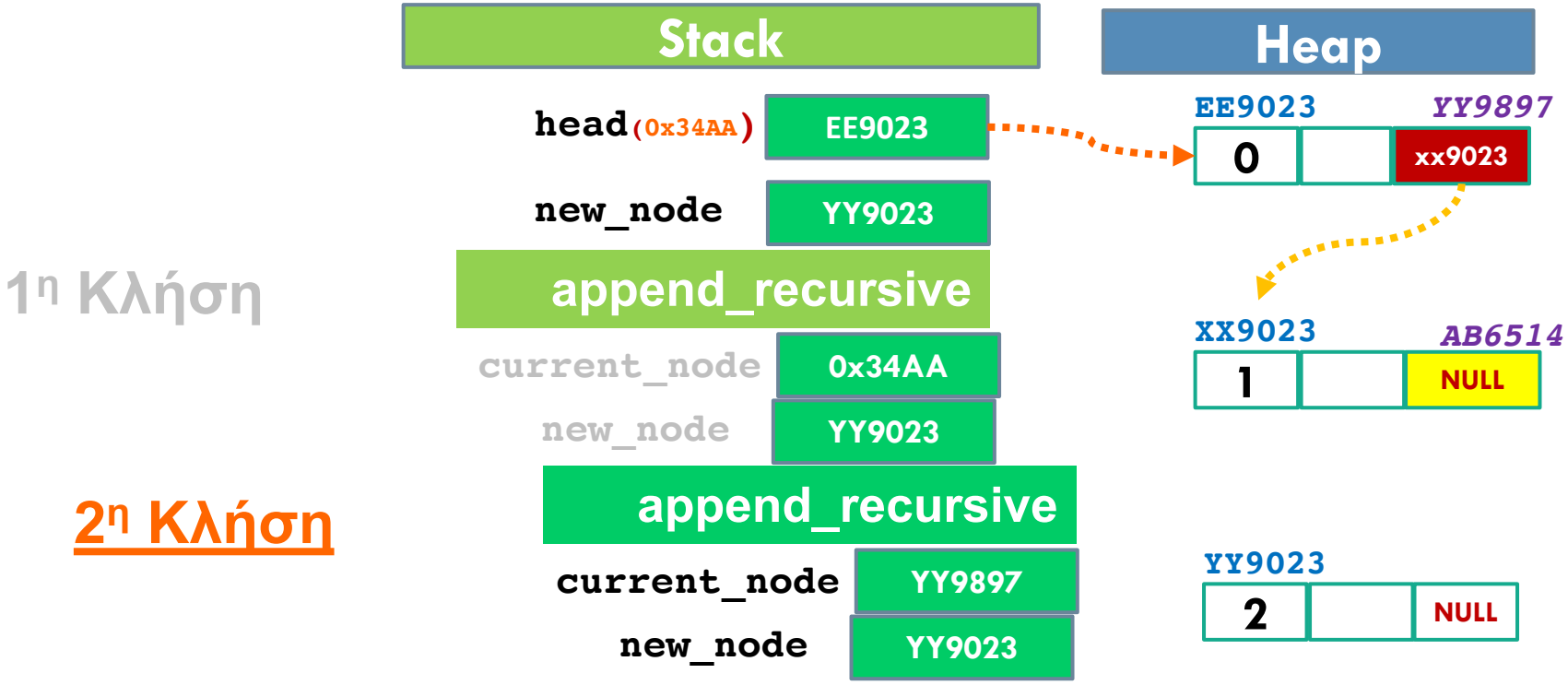

```c
void append_recursive(NODE **,NODE *);
int main()
{
    NODE* head= (NODE *)malloc(sizeof(NODE));
    NODE* new_node;
    int i;
    head->val = 0;
    head->next = NULL;
    for(i=1;i<3;i++){
        new_node = (NODE *)malloc(sizeof(NODE));
        node->val = i;
        node->next = NULL;
        append_recursive(&head, new_node);
     }
    return 0;
}
```

Στις επόμενες διαφάνειες θεωρούμε την εκτέλεση της append_recursive όταν i=2

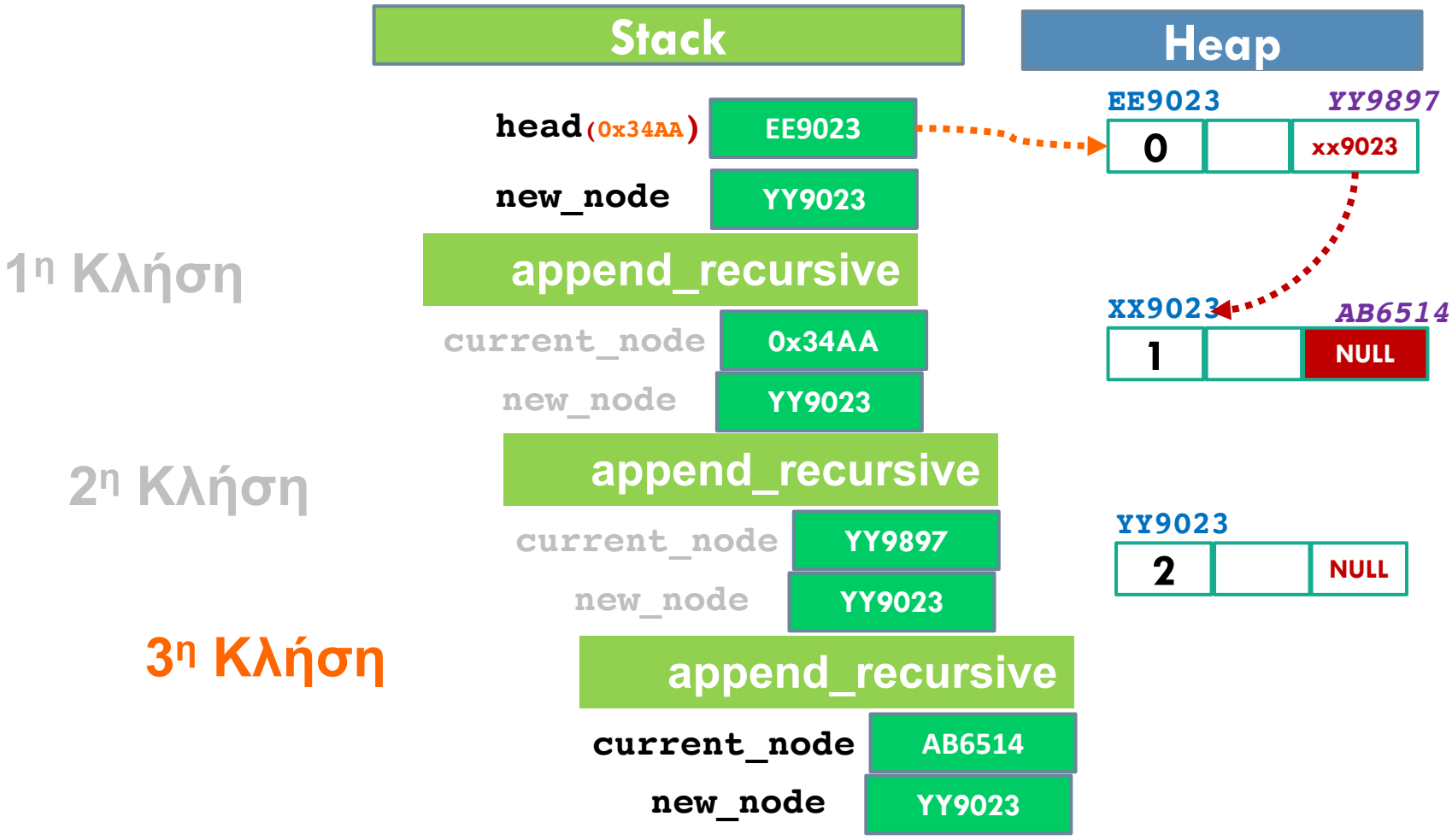


1η Κλήση

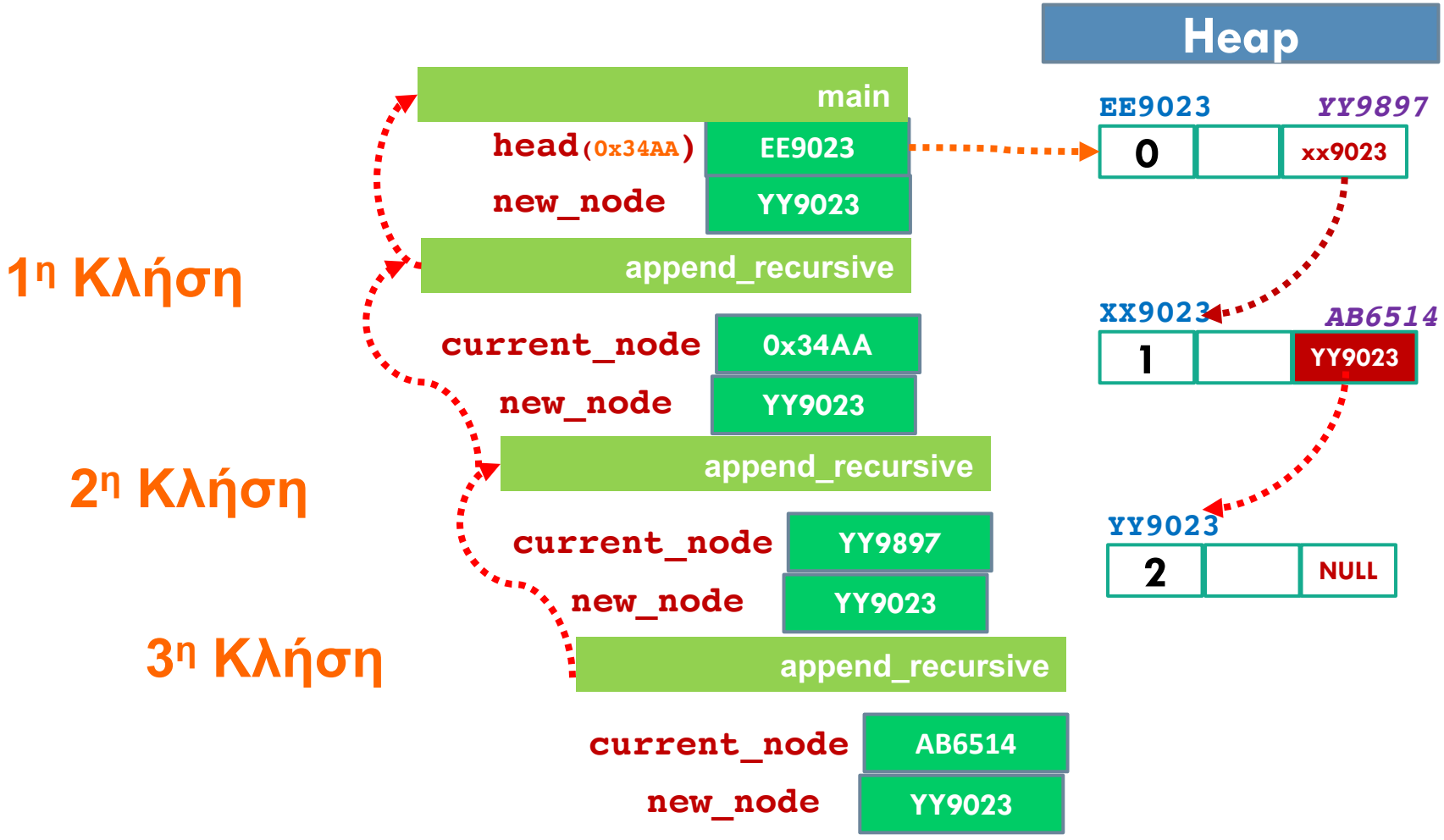
```
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) {
            *current_node = new_node;
            return ;
        }
        append_recursive( & ( (*current_node)->next ), new_node );
        return;
}
```




```
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) {
            *current_node = new_node;
            return ;
        }
        append_recursive( & ( (*current_node)->next ), new_node );
        return;
}
```




```
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) {
              *current_node = new_node;
         return ;
        }
        append_recursive( & ( (*current_node)->next ), new_node );
      return;
}
```




```
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) {
                *current_node = new_node;
                return ;
        }
        append_recursive( & ( (*current_node)->next ), new_node );
      return;
}
```

Ευχαριστώ για την προσοχή σας

Επικοινωνία

- Skype: fidas.christos
- Email: fidas@upatras.gr
- Phone: 2610 – 996491
- Web: http://cfidas.info

Άμεση Επικοινωνία μέσω Skype

SkypeID:
fidas.christos

- Ώρες γραφείου: Τετάρτη & Παρασκεύη 11:00-13:00

Join Zoom Meeting
https://upatras-gr.zoom.us/j/95080297961?pwd=MzRta0JRd3ZwVEVrREZNc09qbG1Zdz09

Το υλικό της διάλεξης είναι διαθέσιμο στο eclass

- https://eclass.upatras.gr/

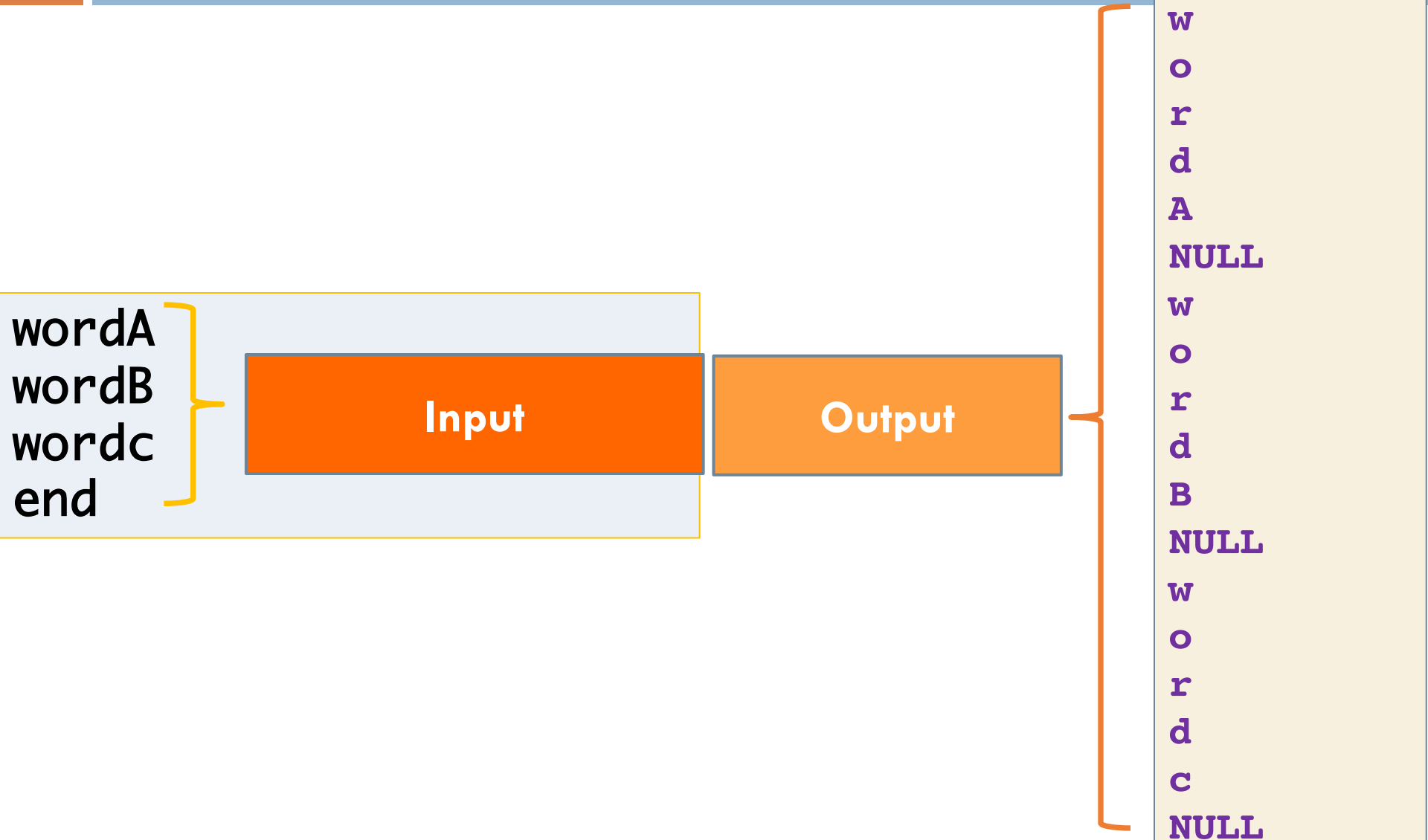
ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

13η Εβδομάδα: Ανασκόπηση Μαθήματος

Άσκηση

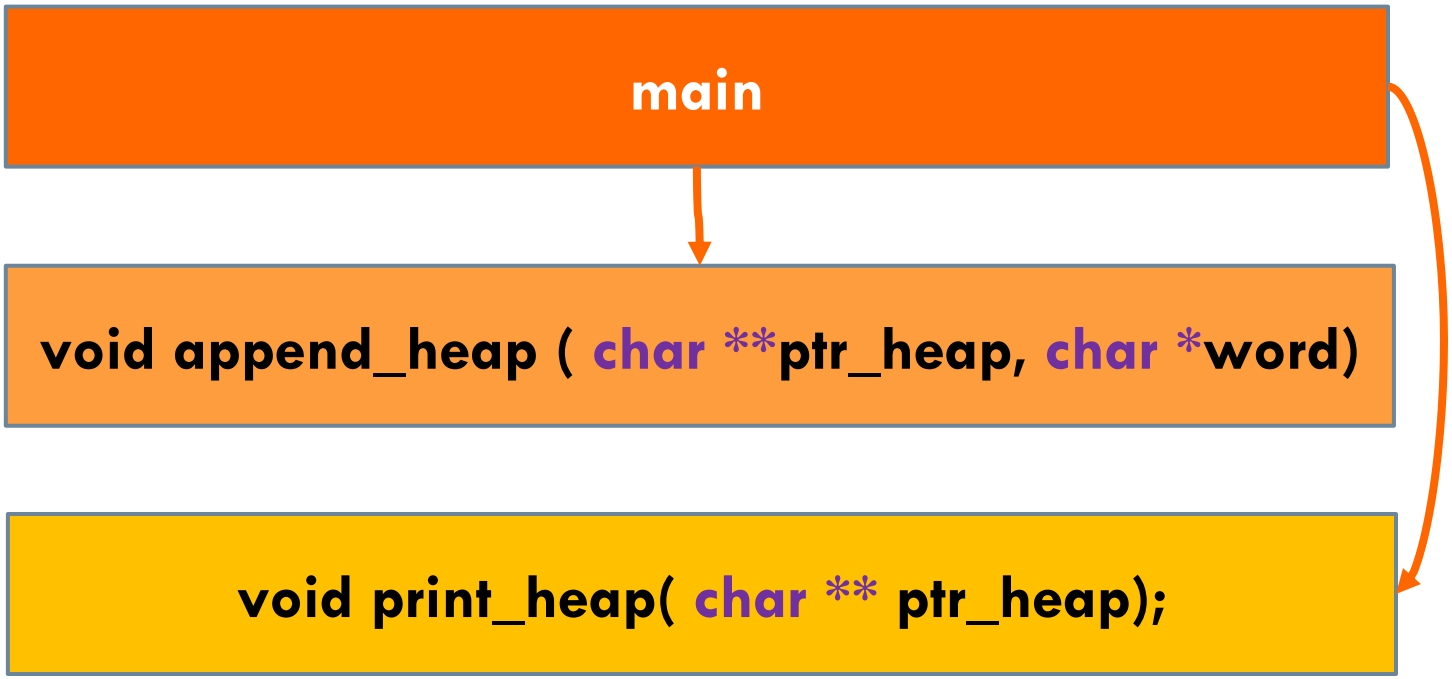
- Να γραφεί ένα πρόγραμμα το οποίο θα ζητά από τον χρήστη να εισάγει απεριόριστο αριθμό συμβολοσειρών από το πληκτρολόγιο (η συνθήκη τερματισμού είναι η εισαγωγή της συμβολοσειράς “end”).
- Στη συνέχεια το πρόγραμμα θα καλεί μια συνάρτηση η οποία χρησιμοποιώντας την realloc θα δεσμεύει χώρο στη σωρό όπου θα αποθηκεύει κάθε χαρακτήρα της κάθε συμβολοσειράς (συμπεριλαμβανομένου του ‘/0’) που έχει εισάγει ο χρήστης.
- Το πρόγραμμα θα εκτυπώνει με χρήση μιας συνάρτησης χαρακτήρα – χαρακτήρα τα περιεχόμενα της μνήμης που δεσμεύτηκαν στη σωρό.

Παράδειγμα Εκτέλεσης



Προσδιορισμός Συναρτήσεων

Τι θα κάνει κάθε συνάρτηση;



Έκδοση 1η

Να γραφεί ένα πρόγραμμα το οποίο θα ζητά από τον χρήστη να εισάγει απεριόριστο αριθμό συμβολοσειρών από το πληκτρολόγιο (η συνθήκη τερματισμού είναι η εισαγωγή της συμβολοσειράς “end”).

```
#include <stdio.h>
#include <string.h>
#define MAX_LENGTH 100
int main()
{
    char word[MAX_LENGTH]={};
    while( scanf("%s",word) ,  strcmp(word, "end")!=0 ){
            printf("%s", word);
    }
   return 0;
}
```

Έκδοση 1η – Παράδειγμα Εκτέλεσης

Να γραφεί ένα πρόγραμμα το οποίο θα ζητά από τον χρήστη να εισάγει απεριόριστο αριθμό συμβολοσειρών από το πληκτρολόγιο (η συνθήκη τερματισμού είναι η εισαγωγή της συμβολοσειράς “end”).

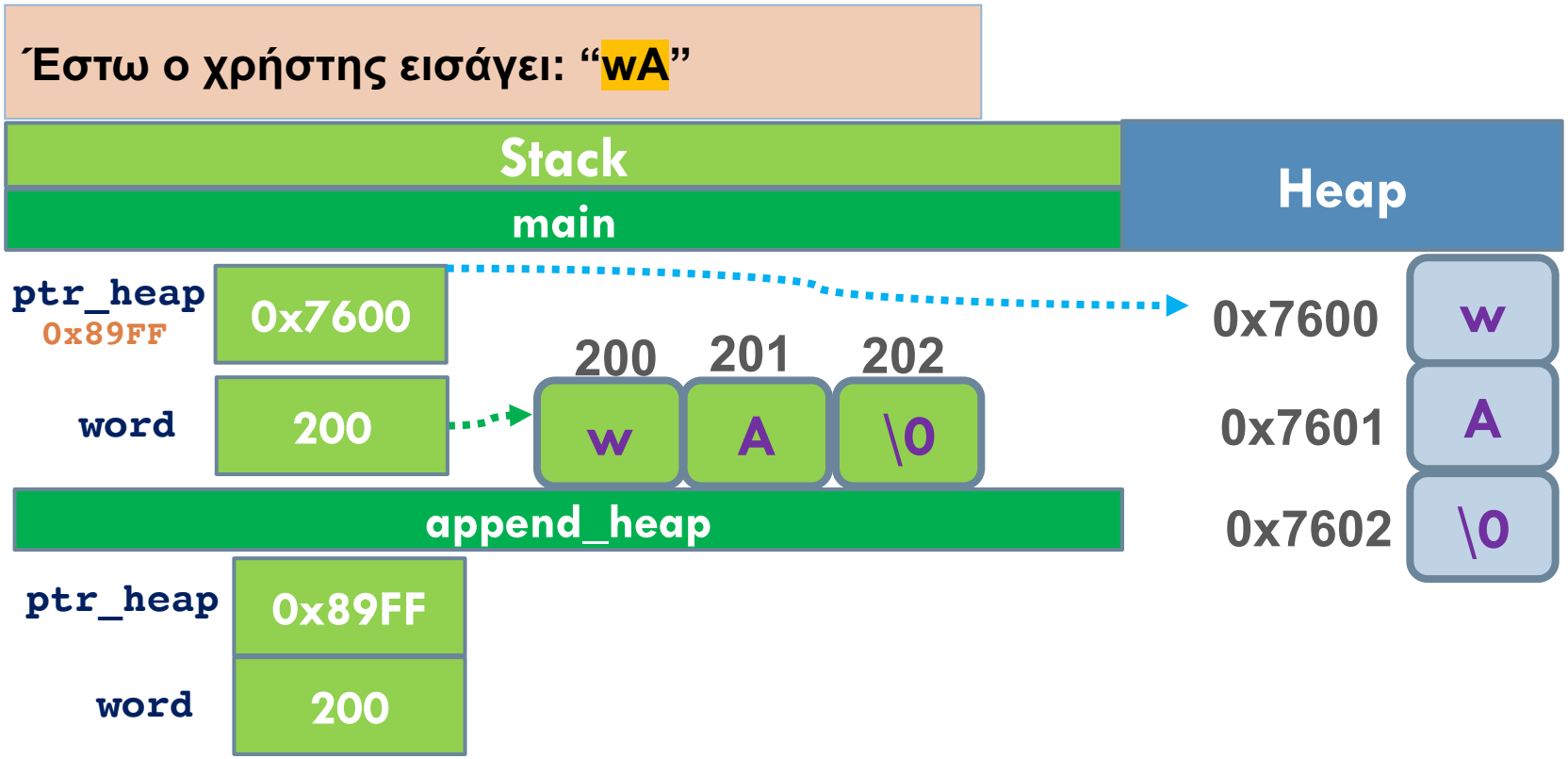
```
#include <stdio.h>
#include <string.h>
#define MAX_LENGTH 100
int main()
{
    char word[MAX_LENGTH]={};
    while( scanf("%s",word) ,  strcmp(word, "end")!=0 ){
            printf("%s", word);
    }
   return 0;
}
```

```
wordA
wordA
wordB
wordB
wordc
wordc
end

Process returned 0 (0x0)   execution time : 14.451 s
Press ENTER to continue.
```

Έκδοση 2η – Διαχείριση Σωρού

- Στη συνέχεια το πρόγραμμα θα καλεί μια συνάρτηση η οποία χρησιμοποιώντας την realloc θα δεσμεύει χώρο στη σωρό όπου θα αποθηκεύει κάθε χαρακτήρα τη κάθε συμβολοσειράς που εισάγει ο χρήστης.

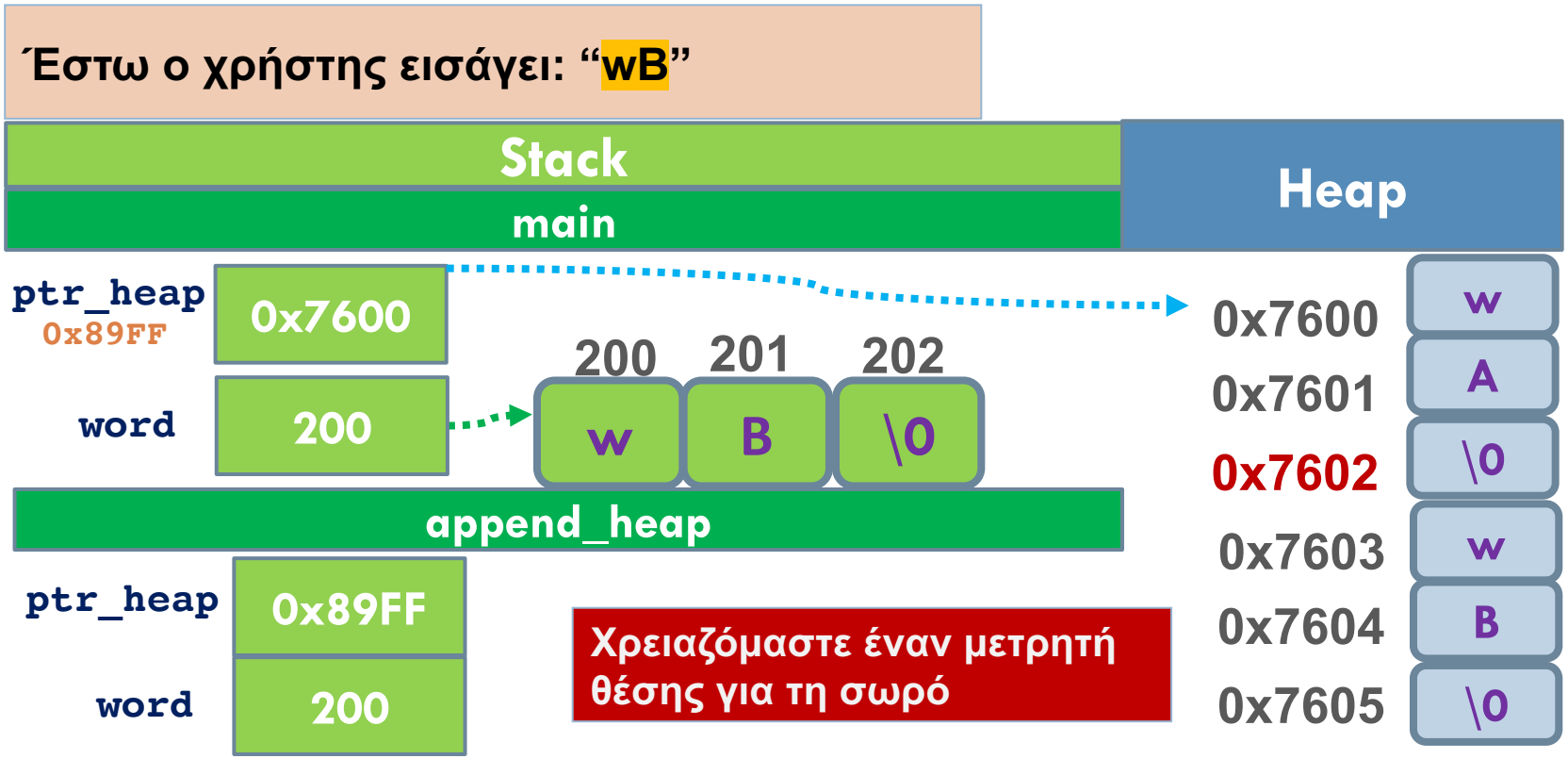
void append_heap (char **ptr_heap, char *word)



Έκδοση 2η – Διαχείριση Σωρού

- Στη συνέχεια το πρόγραμμα θα καλεί μια συνάρτηση η οποία χρησιμοποιώντας την realloc θα δεσμεύει χώρο στη σωρό όπου θα αποθηκεύει κάθε χαρακτήρα τη κάθε συμβολοσειράς που εισάγει ο χρήστης.

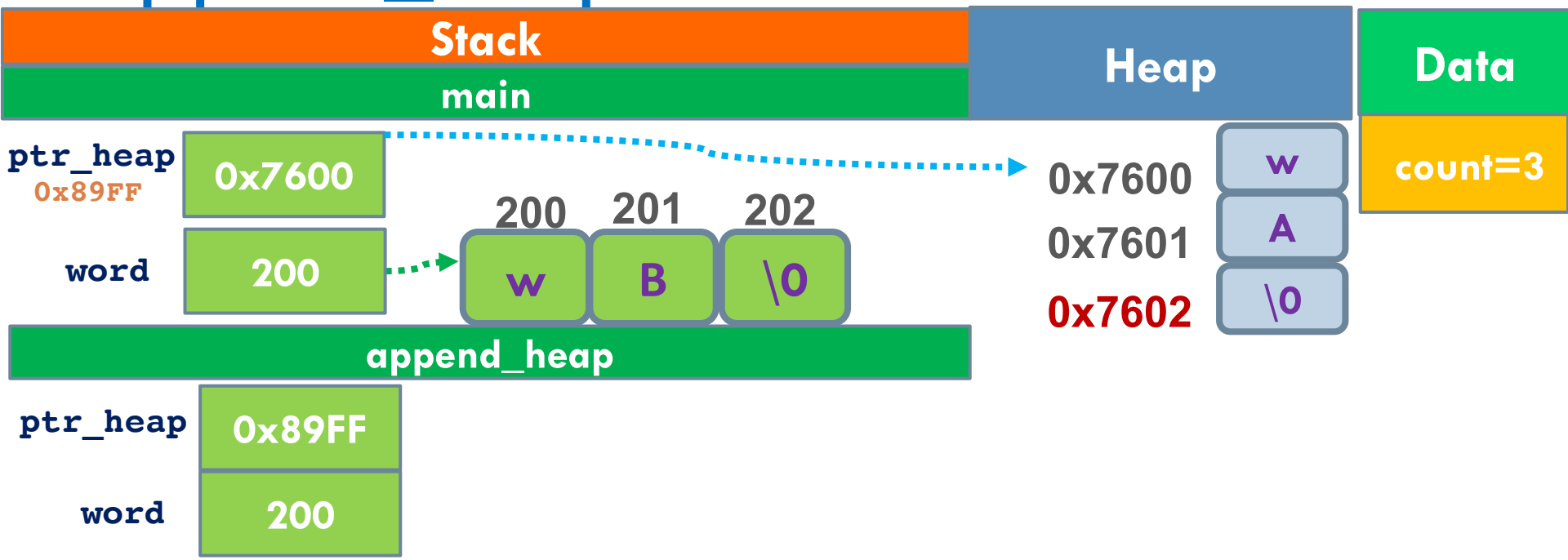
void append_heap (char **ptr_heap, char *word)



```c
#include <stdio.h>
#include <string.h>
#include <string.h>
#define MAX_LENGTH 100
void append_heap (char **, char *);
int count=0;
int main()
{
    char word[MAX_LENGTH]={};
    char *ptr_heap=NULL;
    while( scanf("%s",word) ,  strcmp(word, "end")!=0 ){
          append_heap(&ptr_heap, word);
    }
   free(ptr_heap);
   return 0;
}
```

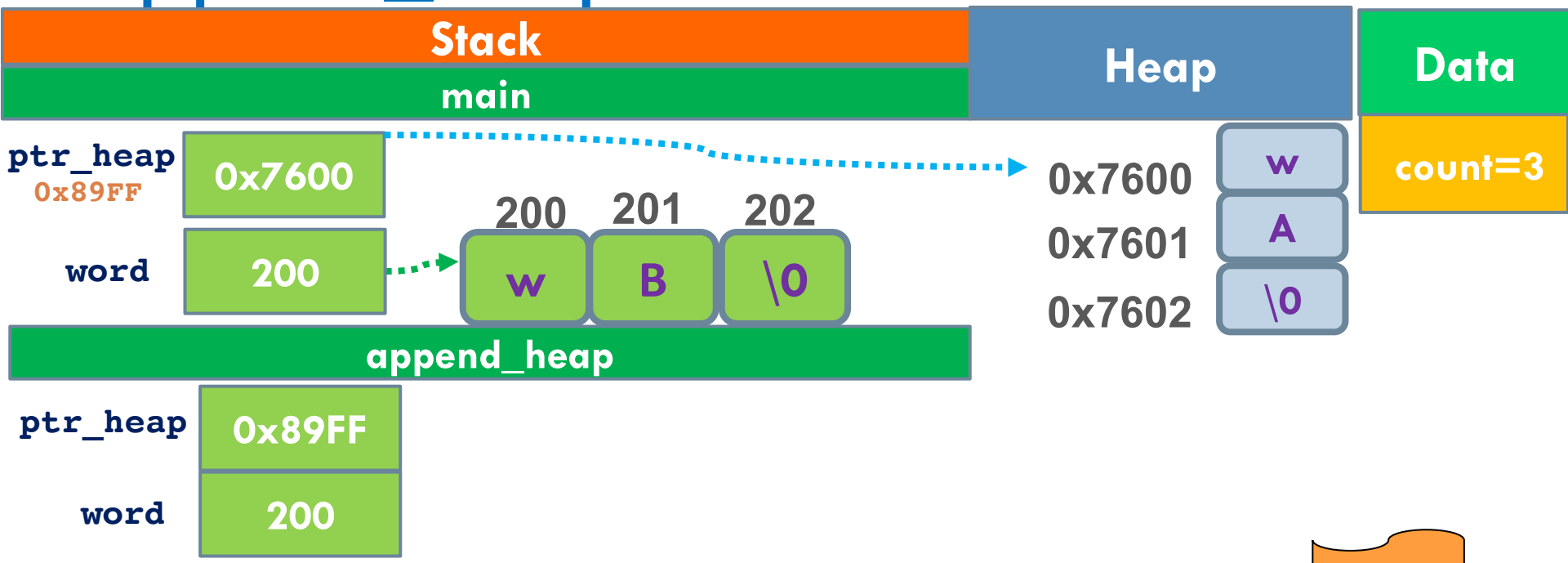
```c
void append_heap(char **ptr_heap, char *word){
  *ptr_heap=realloc(*ptr_heap, (strlen(word)+1+count) * sizeof (char) );
   while(*word!='\0')
         *(*ptr_heap+count++)=*(word++);

  *(*ptr_heap+count++)='\0';
}
```

H append_heap




```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) + 1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

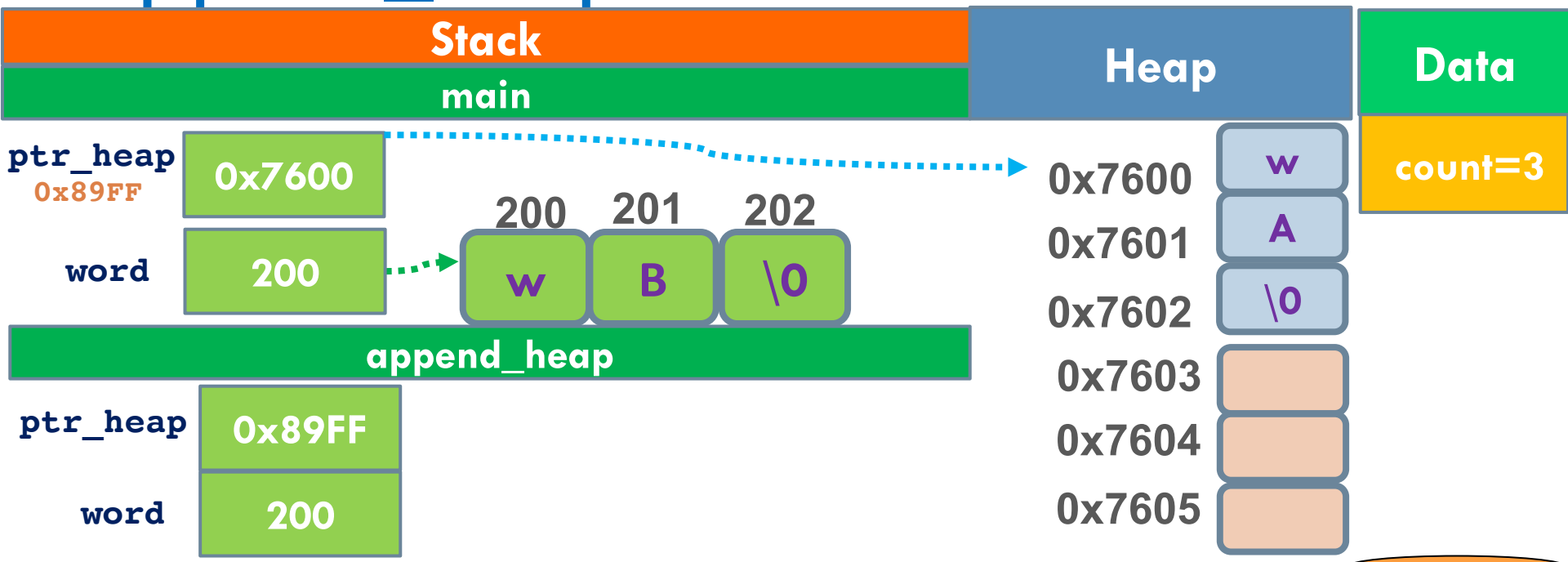
H append_heap



size=3

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

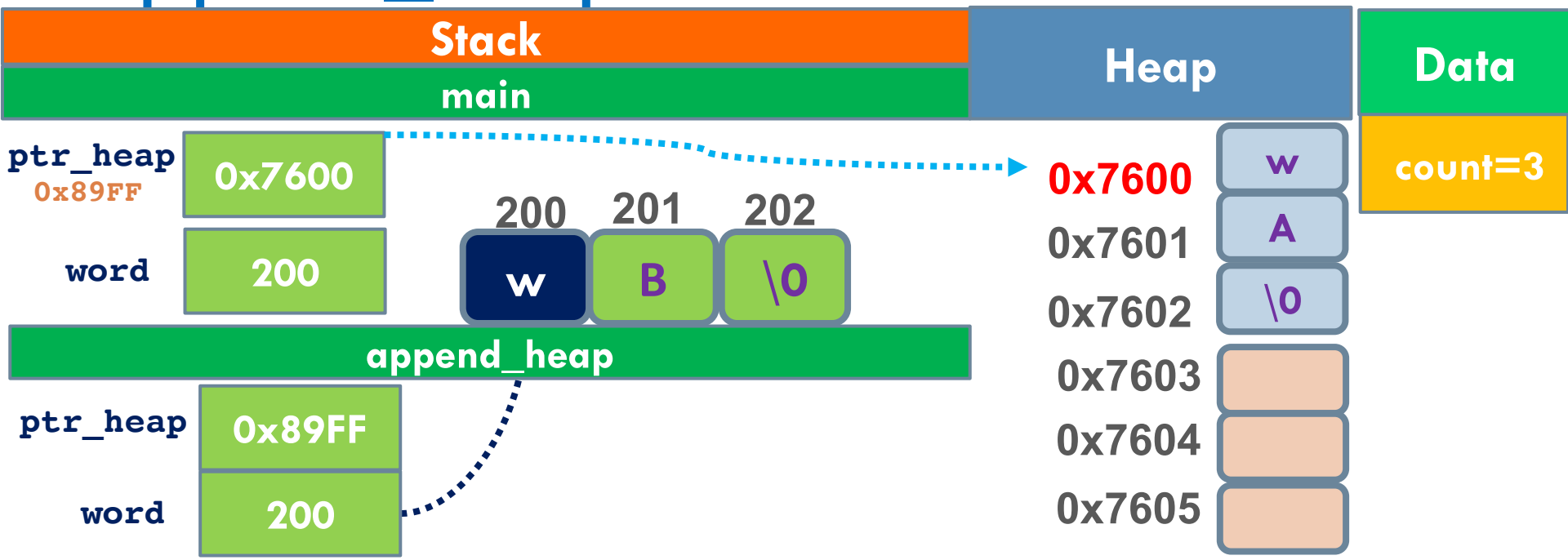
H append_heap




```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

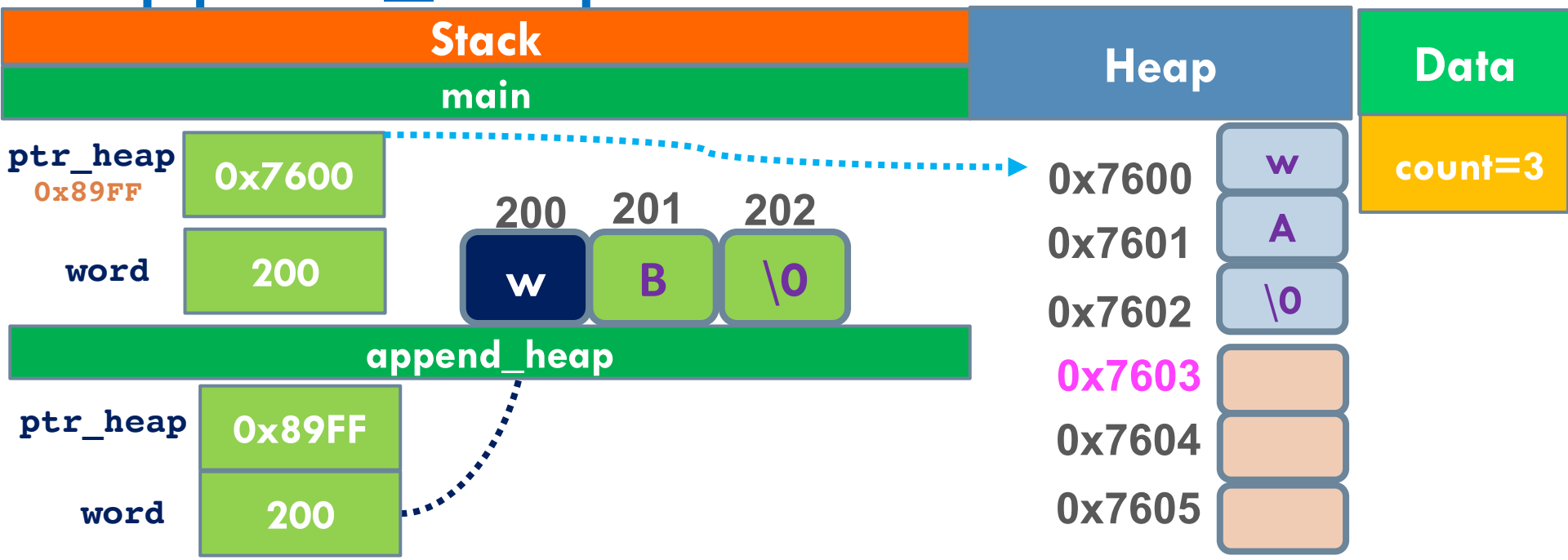
*ptr_heap=0x7600

H append_heap



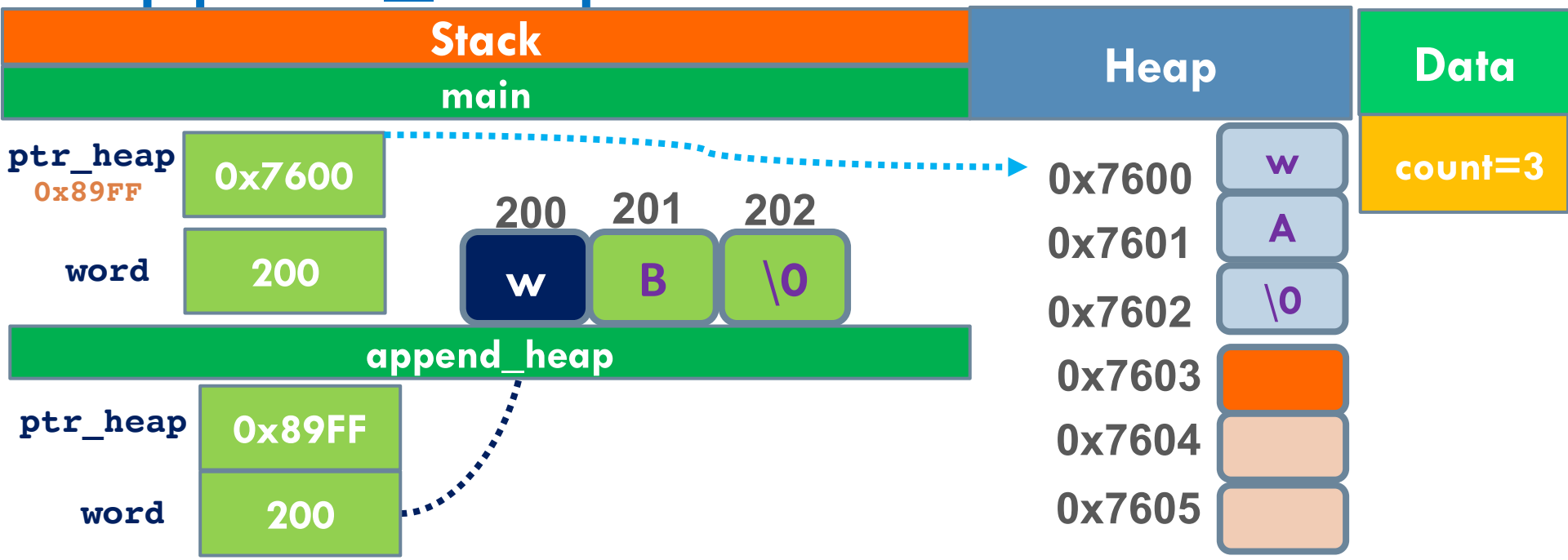

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



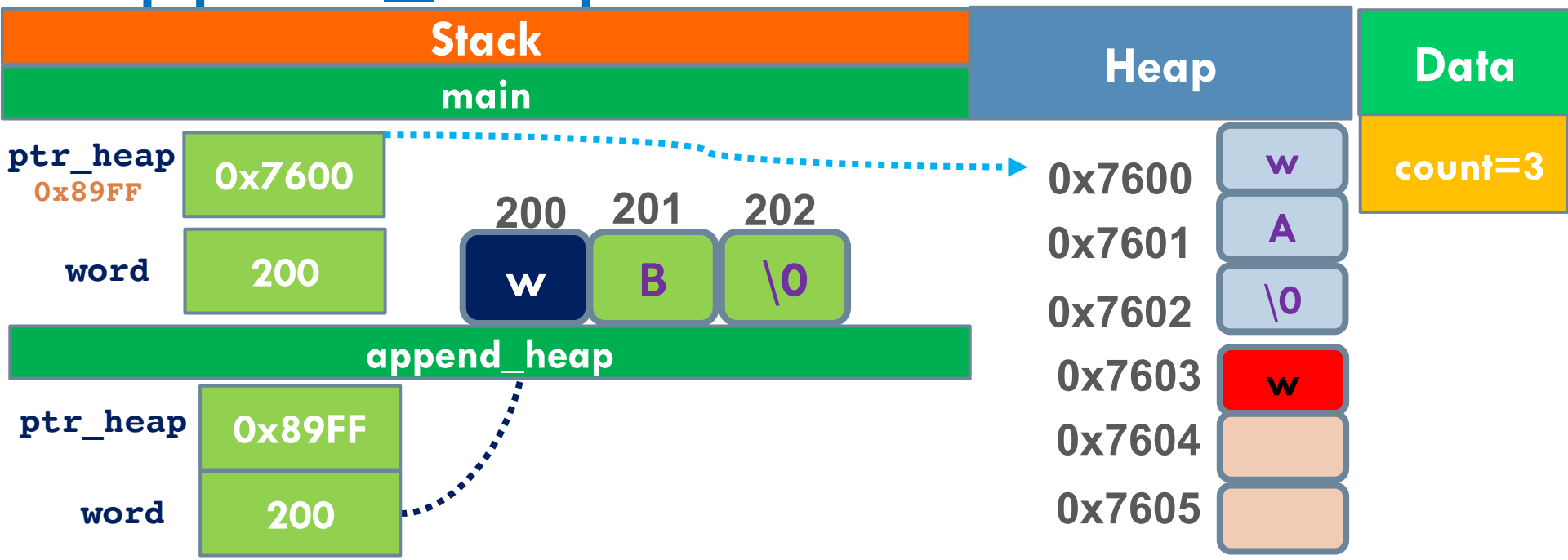

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



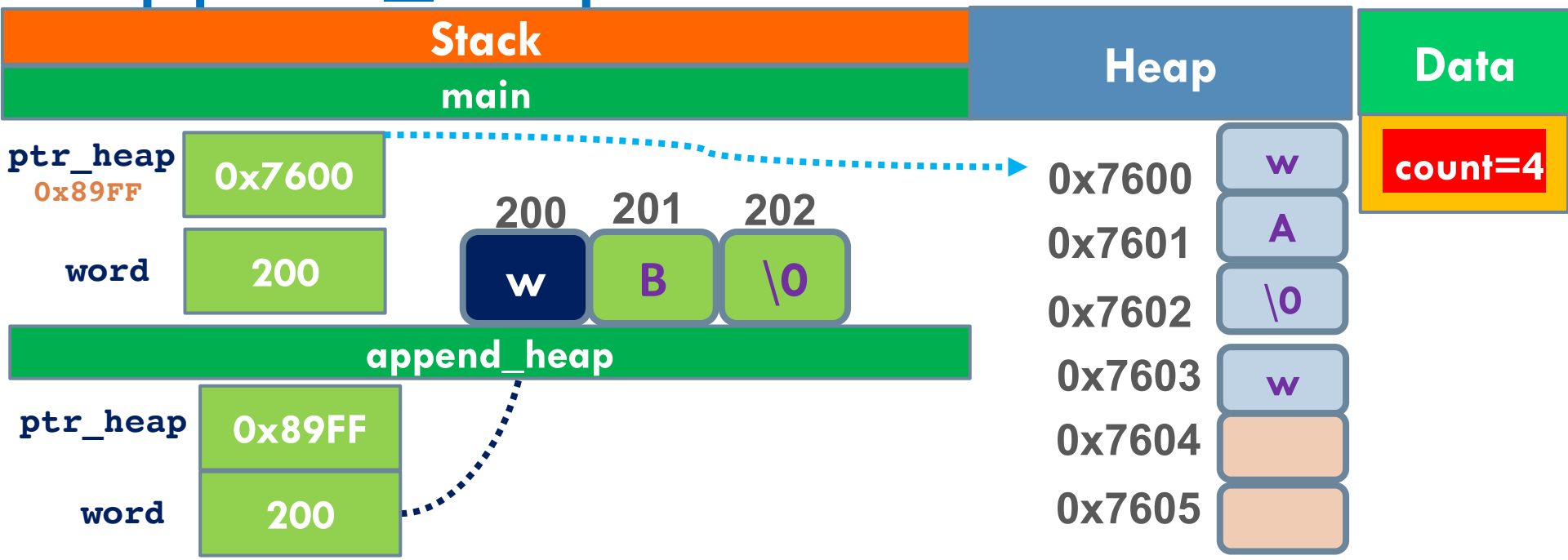

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



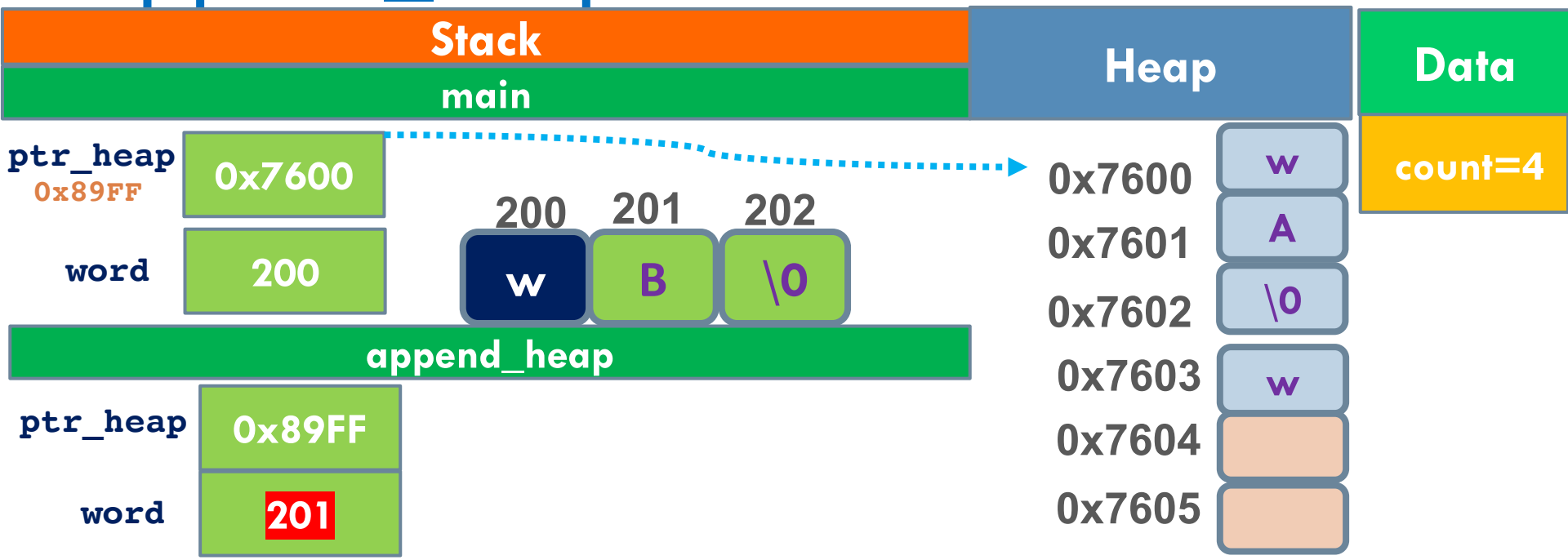

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



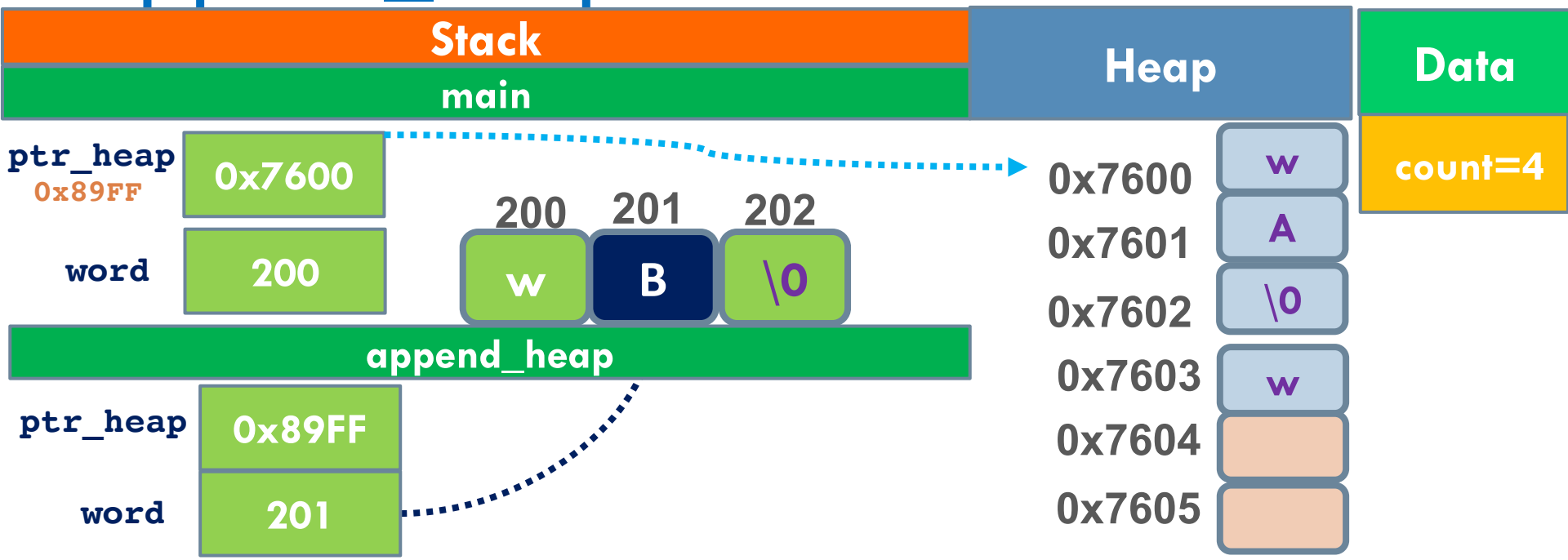

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



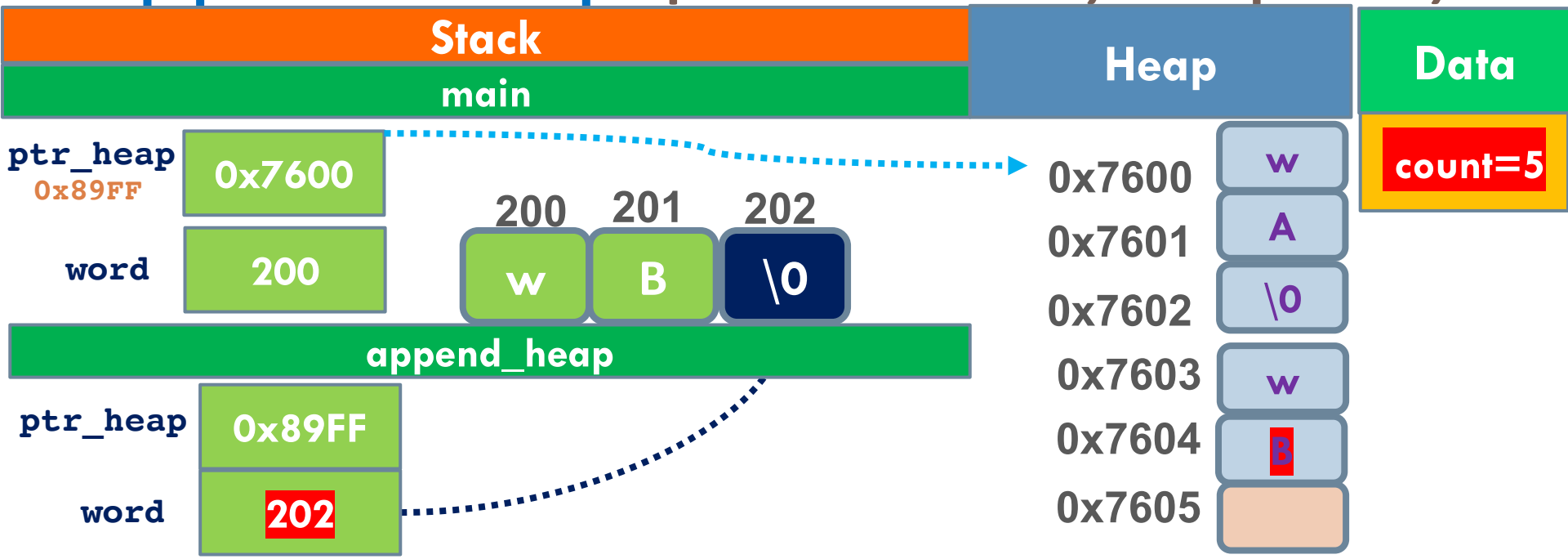

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



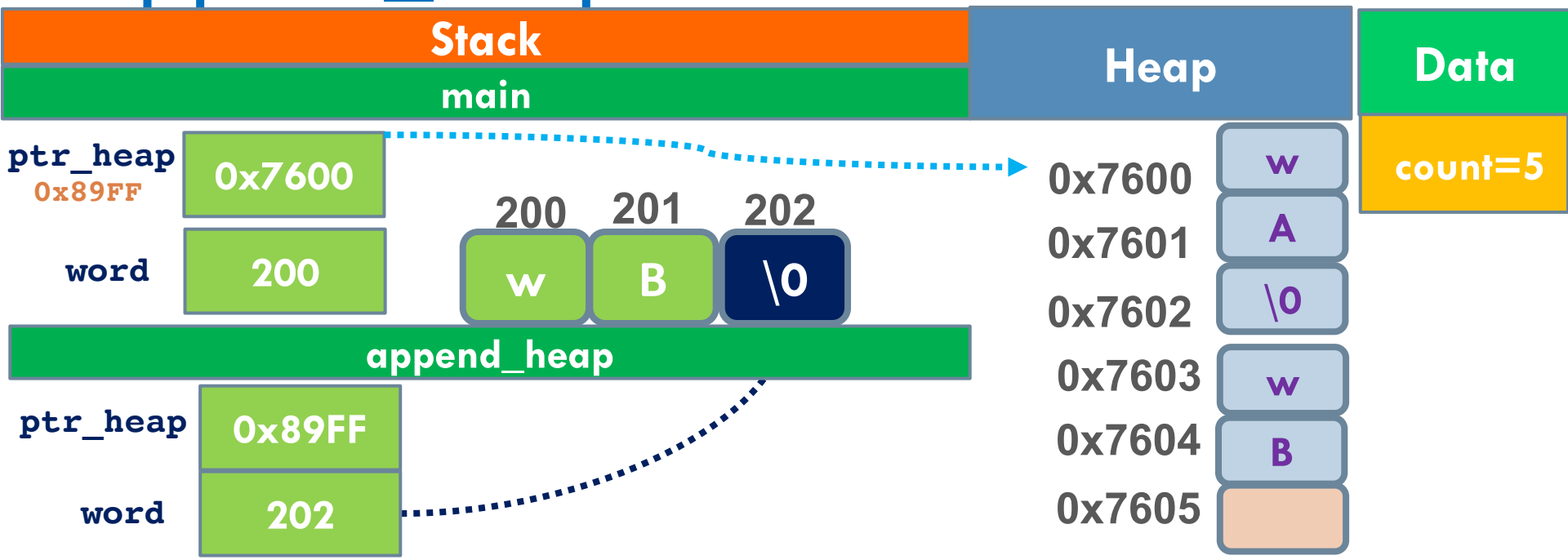

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size + count);
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

Η append_heap για κοινούς θνητούς



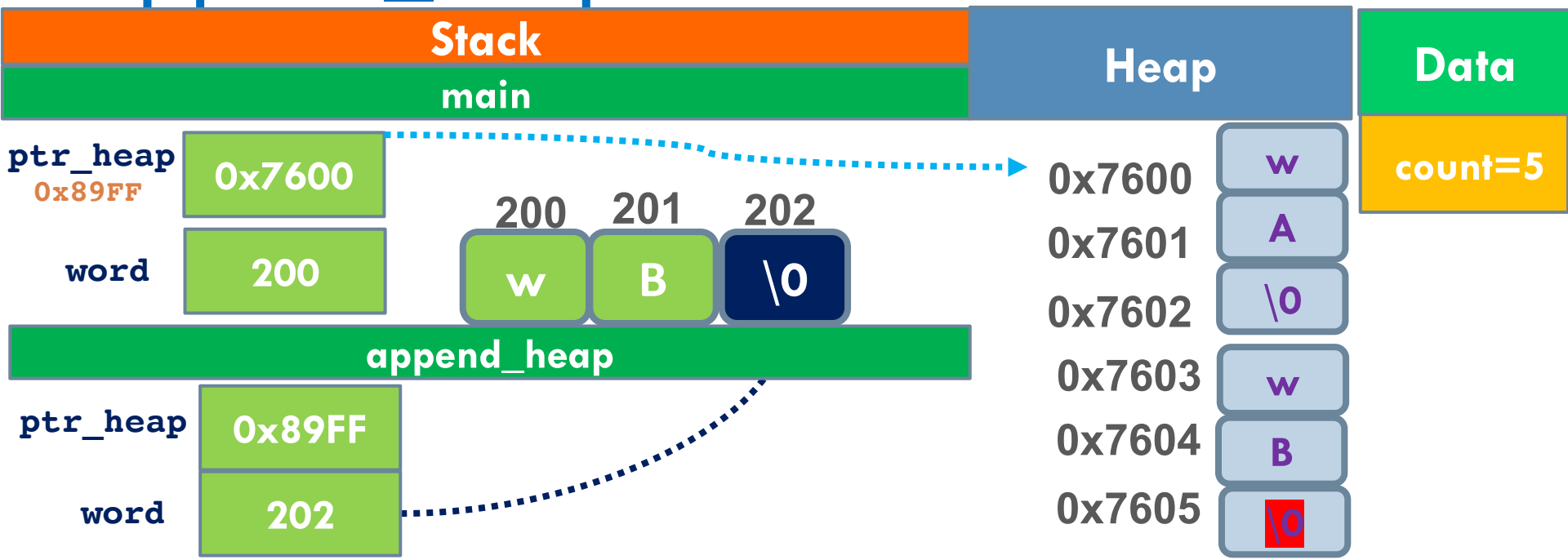

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size + count );
        if(*ptr_heap==NULL) exit(1);
       while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap



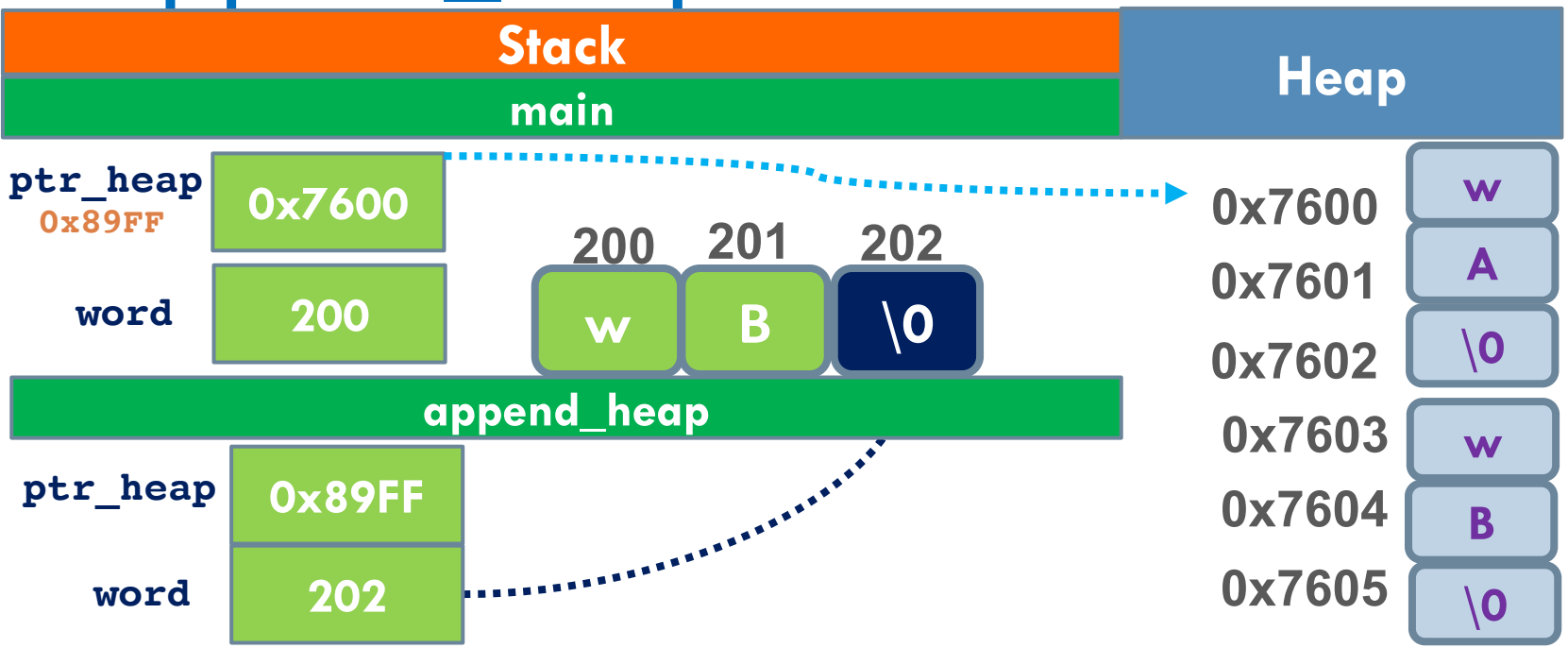

```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
       while(*word!='\0'){
            *(*ptr_heap+count)=*word;
             count++;
             word++;
         }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap




```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
       while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

H append_heap

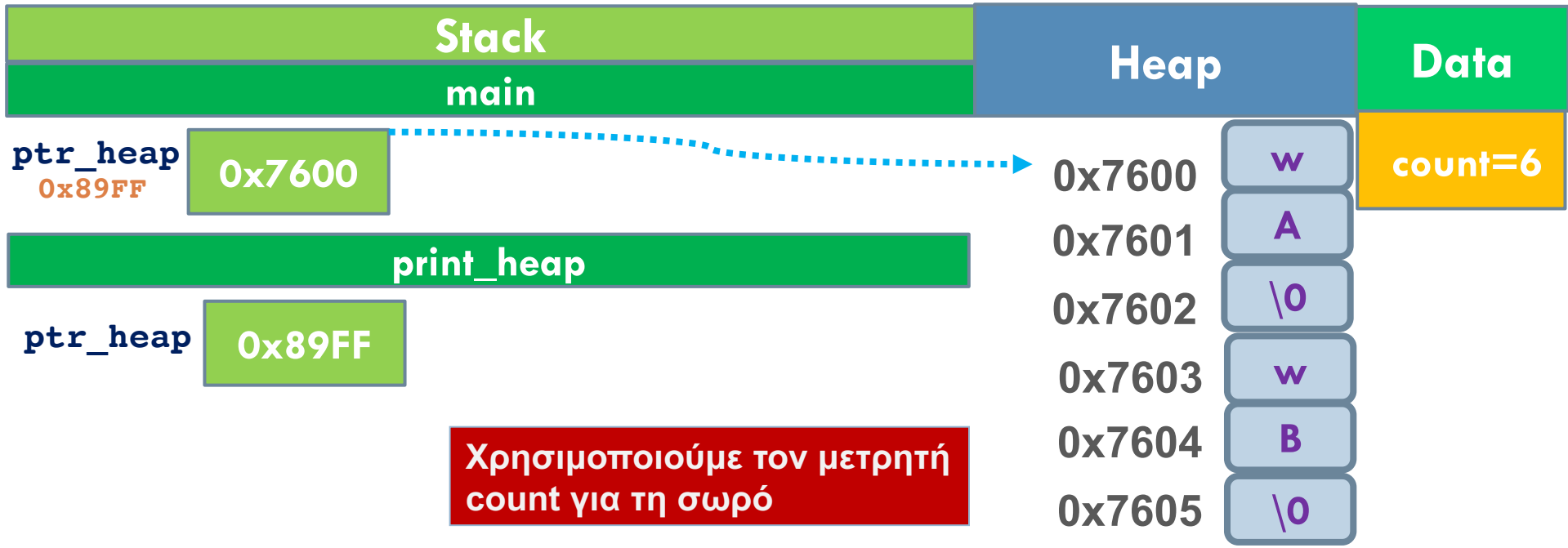



```
void append_heap(char **ptr_heap, char *word){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size+count );
        if(*ptr_heap==NULL) exit(1);
       while(*word!='\0'){
            *(*ptr_heap+count)=*word;
            count++;
            word++;
        }
       *( *ptr_heap+count)='\0';
       count++;
}
```

Έκδοση 3η – Εκτύπωση Σωρού

- Το πρόγραμμα θα εκτυπώνει με χρήση μιας συνάρτησης χαρακτήρα – χαρακτήρα τα περιεχόμενα της μνήμης που δεσμεύτηκαν στη σωρό.

void print_heap (char **ptr_heap)



Έκδοση 3η – Εκτύπωση Σωρού

- Στη συνέχεια το πρόγραμμα θα καλεί μια συνάρτηση η οποία χρησιμοποιώντας την realloc θα δεσμεύει χώρο στη σωρό όπου θα αποθηκεύει κάθε χαρακτήρα τη κάθε συμβολοσειράς που εισάγει ο χρήστης.

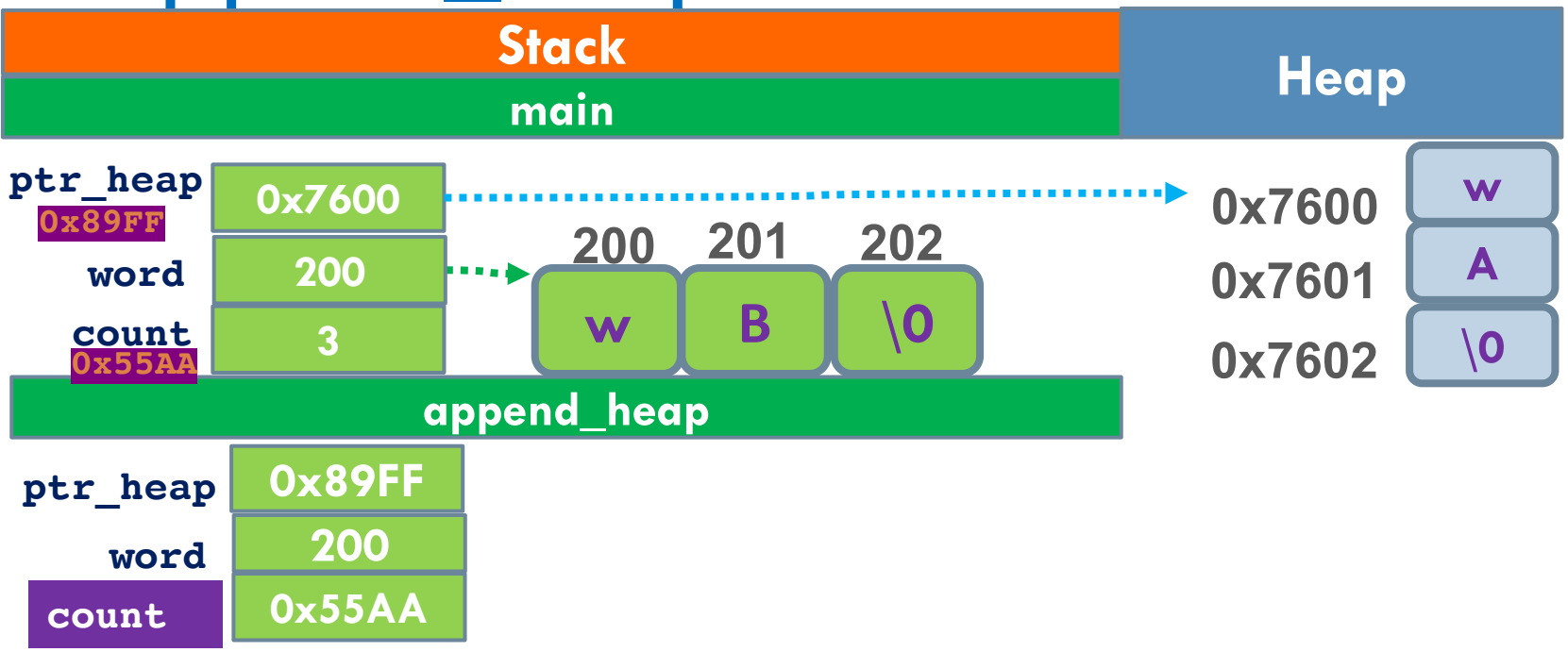
```
void print_heap(char **ptr){
     int k=0;
     char x;
     printf("HEAP:%d\n", count);
     while(k<count){
        x=(*(*ptr+k));
        if(x=='\0') printf("NULL\n");
        else printf("%c\n", (*(*ptr+k)));
        k++;
     }
}
```

Διαφορετική Έκδοση – Χωρίς τη χρήση γενερικής μεταβλητής **count**

```c
#include <stdio.h>
#include <string.h>
#include <string.h>
#define MAX_LENGTH 100
void append_heap (char **, char *, int *);
void print_heap(char*, int *);
int main(){
    int count=0;
    char word[MAX_LENGTH]={};
    char *ptr_heap=NULL;
    while( scanf("%s",word) ,  strcmp(word, "end")!=0 ){
            append_heap(&ptr_heap, word, &count);
    }
    free(ptr_heap);
    return 0;
}
```

```c
void append_heap(char **ptr_heap, char *word, int *count){
    int size=(strlen(word) +1) * sizeof (char);
    *ptr_heap=(char *) realloc(*ptr_heap, size+ (*count) );
    if(*ptr_heap==NULL) exit(1);
    while(*word!='\0'){
        *(*ptr_heap+ (*count) )=*word;
        *count=*count+1;
        word++;
    }
   *( *ptr_heap+ (*count) )='\0';
   *count=*count+1;
}
```

H append_heap




```
void append_heap(char **ptr_heap, char *word, int *count){
        int size=(strlen(word) +1) * sizeof (char);
        *ptr_heap=(char *) realloc(*ptr_heap, size + *count);
        if(*ptr_heap==NULL) exit(1);
        while(*word!='\0'){
            *(*ptr_heap+ (*count) )=*word;
            *count=*count+1;
            word++;
        }
       *( *ptr_heap+ (*count) )='\0';
       *count=*count+1;
}
```

Η print_heap (με απλό δείκτη)

```
#include <stdio.h>
#include <string.h>
#include <string.h>
#define MAX_LENGTH 100
void append_heap (char **, char *, int *);
void print_heap(char*, int *);
int main(){
    int count=0;
    char word[MAX_LENGTH]={};
    char *ptr_heap=NULL;
    while( scanf("%s",word) ,  strcmp(word, "end")!=0 ){
            append_heap(&ptr_heap, word, &count);
    }
    print_heap(ptr_heap,&count);
    free(ptr_heap);
    return 0;
}
```

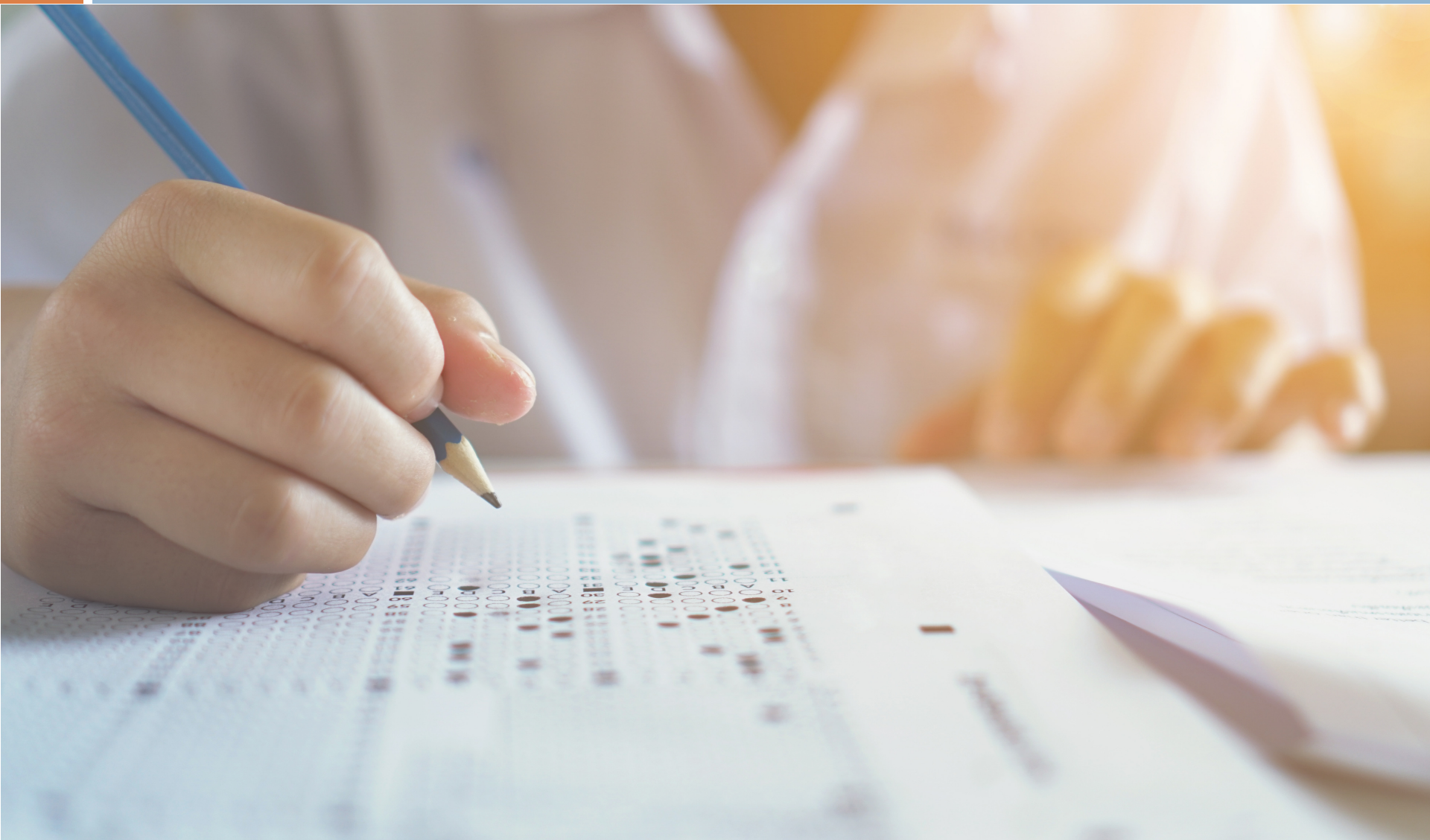
```
void print_heap(char *ptr, int *count){
    int k=0;
    char x;
    printf("HEAP:%d\n", *count);
    while(k<*count){
      x=(*(ptr+k));
      if(x=='\0') printf("NULL\n");
      else printf("%c\n", (*(ptr+k)));
      k++;
    }
   return;
}
```

Διαφορετική Έκδοση – Χρήση static count στη συνάρτηση append_heap

```c
#include <stdio.h>
#include <string.h>
#include <string.h>
#define MAX_LENGTH 100
void print_heap(char *, int *count);
int append_heap(char **ptr_heap, char *word);

int main()
{
     int count=0;
     char word[MAX_LENGTH]={};
     char *ptr_heap=NULL;
     while( scanf("%s",word) ,  strcmp(word, "end")!=0 ){
             count=append_heap(&ptr_heap, word);
     }
     print_heap(ptr_heap,&count);
    free(ptr_heap);
     return 0;
}
```

```c
int append_heap(char **ptr_heap, char *word){
    static int count;
      int size=(strlen(word) +1) * sizeof (char);
     *ptr_heap=(char *) realloc(*ptr_heap, size + count);
       if(*ptr_heap==NULL) exit(1);
     while(*word!='\0'){
        *( *ptr_heap+(count) )=*word;
        count=count+1;
        word++;
      }
     *( *ptr_heap+count)='\0';
     count=count+1;

     return count;
}
```

Ευχές για καλή εξεταστική!



Ευχαριστώ για την προσοχή σας

Επικοινωνία

- Skype: fidas.christos
- Email: fidas@upatras.gr
- Phone: 2610 – 996491
- Web: http://cfidas.info

Άμεση Επικοινωνία μέσω Skype

SkypeID:
fidas.christos

- Ώρες γραφείου: Τετάρτη & Παρασκεύη 11:00-13:00

Join Zoom Meeting
https://upatras-gr.zoom.us/j/95080297961?pwd=MzRta0JRd3ZwVEVrREZNc09qbG1Zdz09

Το υλικό της διάλεξης είναι διαθέσιμο στο eclass

- https://eclass.upatras.gr/