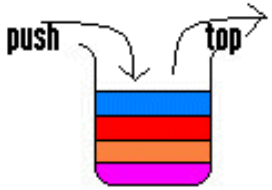
ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

11η Εβδομάδα: Στοίβες, Ουρές και Αναδρομή

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.**

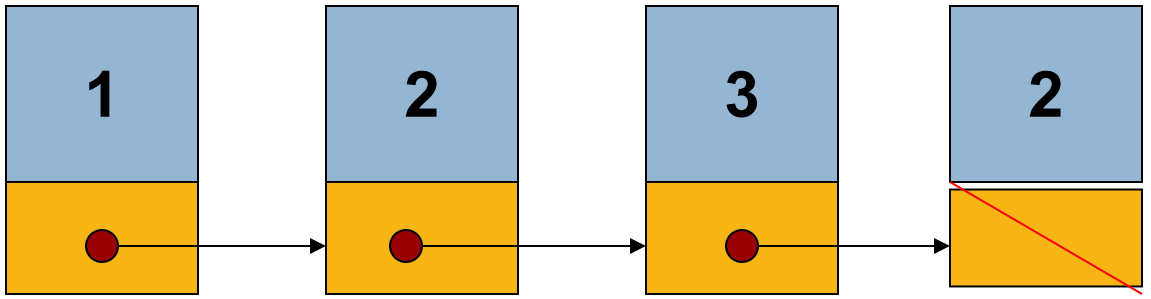
Στοίβες - stacks



- Για την παράσταση μιας στοίβας με στοιχεία α_1, α_2, ..., α_n μπορούμε να χρησιμοποιήσούμε **μια συνδεδεμένη λίστα από κόμβους**.
- Κάθε κόμβος αποτελείται από ένα στοιχείο (στοιχεία της στοίβας) και από ένα δείκτη (προς τον επόμενο κόμβο της στοίβας). Η κορυφή της στοίβας είναι ο πρώτος κόμβος της λίστας.
- Χρησιμοποιούμε μια μεταβλητή για να φυλάγουμε στοιχεία σχετικά με τη στοίβα π.χ. **μέγεθος** (size) και δείκτη προς την **κορυφή της στοίβας (head).**

Στοίβες - stacks

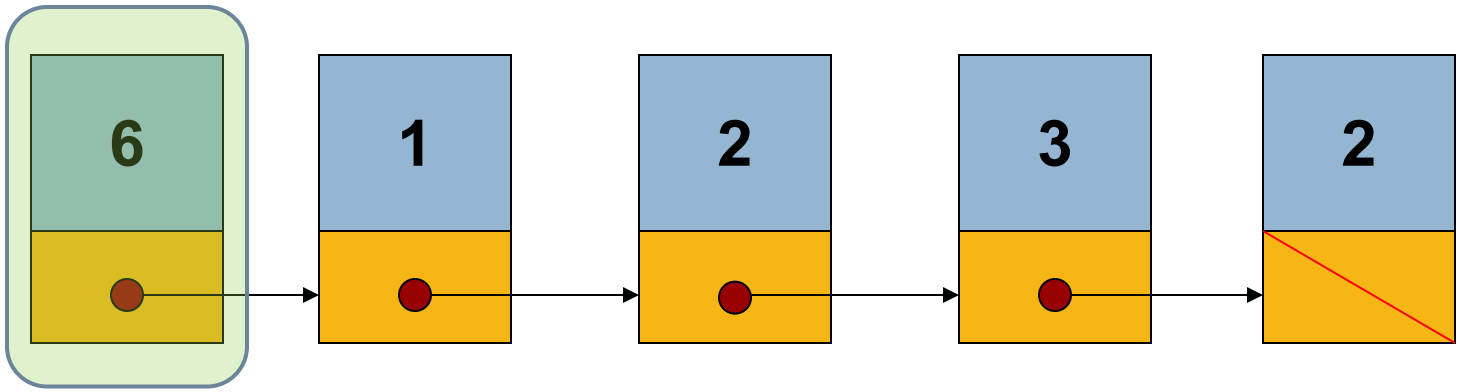
Η υλοποίηση των βασικών λειτουργιών στοίβας μπορεί να γίνει με απλές συνδεδεμένες λίστες.



- Λειτουργίες **push**/**pop** / **top**
- Last-In First-Out (LIFO)

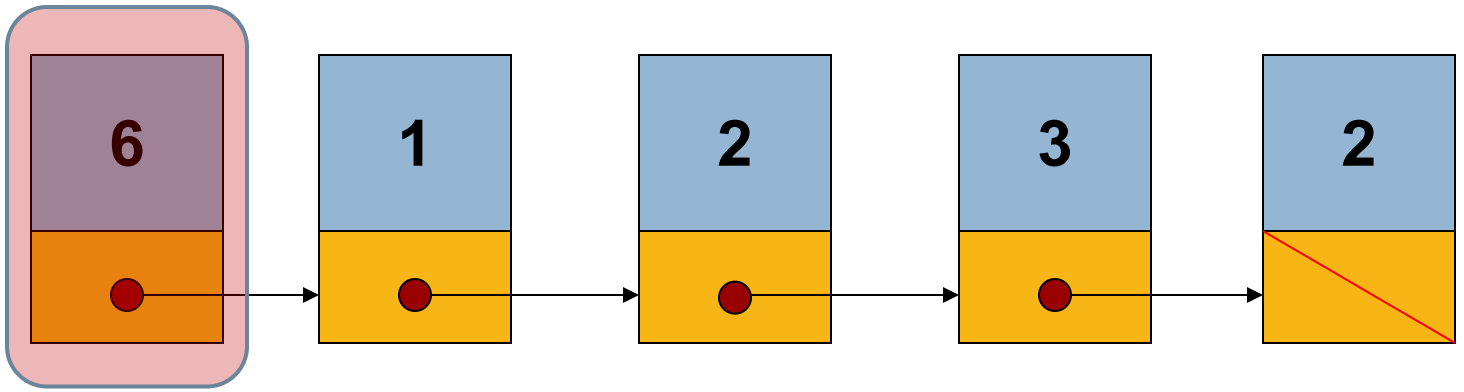
Βασική προϋπόθεση: όλες οι πράξεις να εκτελούνται χωρίς αναζήτηση οποιοδήποτε στοιχείου.

Στοίβες – stacks: push



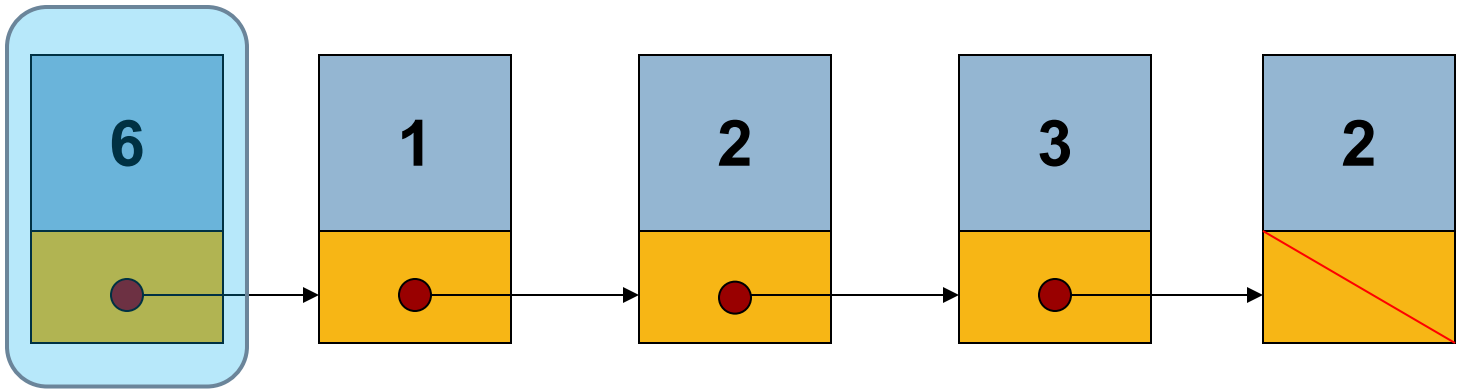
- **push:** Προσθήκη καινούριο κόμβου στην κορυφή της στοίβας.

Στοίβες – stacks: pop



- **pop:** Διαγραφή του κόμβου κορυφής της στοίβας.

Στοίβες – stacks: top



- **top:** Επιστροφή του κόμβου κορυφής της στοίβας.

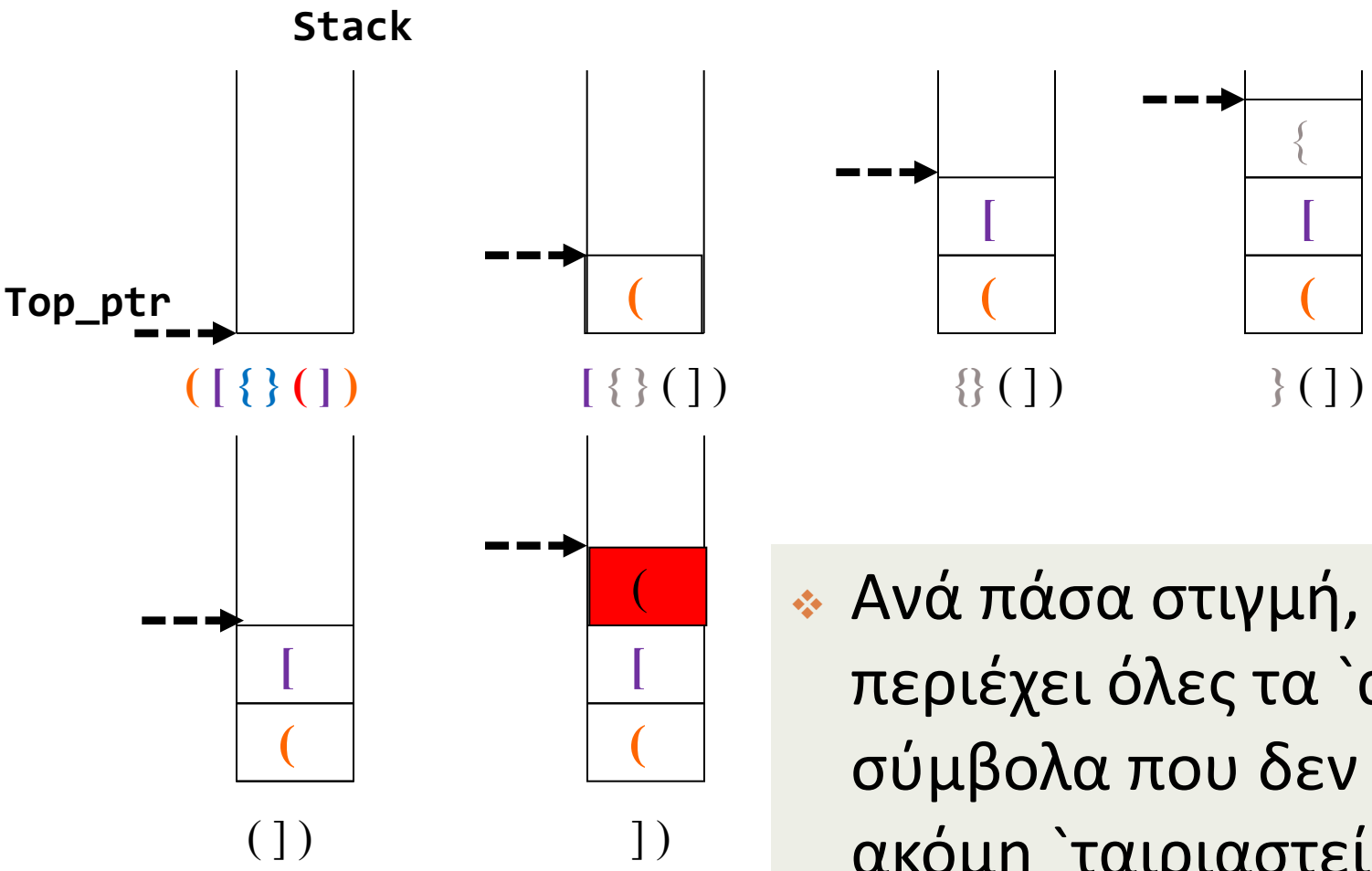
Εφαρμογή : Ισοζυγισμός Συμβόλων

- Ο έλεγχος σύνταξης (π.χ. ενός προγράμματος) απαιτεί να ταιριάξουμε σύμβολα όπως: (με) ή { με }

- Πρόβλημα: να διαπιστώσετε αν μια συμβολοσειρά που περιέχει τους πιο πάνω χαρακτήρες είναι ισοζυγισμένη, δηλαδή όλα τα σύμβολα ταιριάζουν.

- π.χ. {[]}
  ([{}{}[]])
  ([]{()})

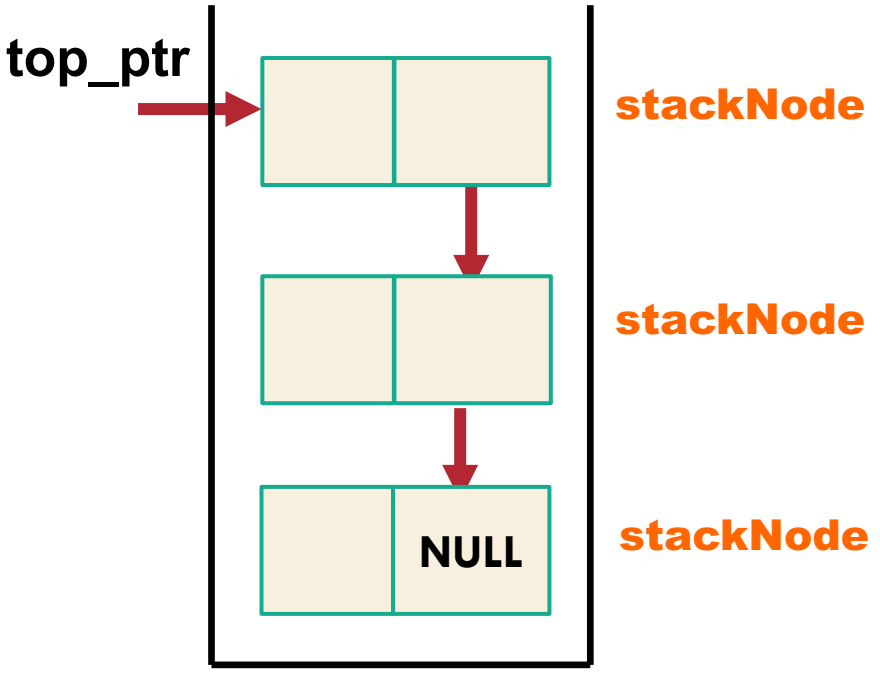
Παράδειγμα Εκτέλεσης



- Ανά πάσα στιγμή, η στοίβα περιέχει όλες τα `αριστερά΄ σύμβολα που δεν έχουν ακόμη `ταιριαστεί΄.

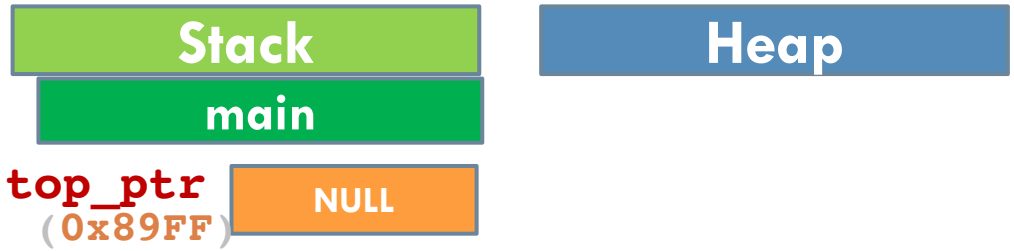
Στοίβα – Stack Υλοποίηση με συνδεδεμένη λίστα

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



- Η υλοποίηση των βασικών πράξεων στοίβας για αναπαράσταση με συνδεδεμένες λίστες.
  - Βασική προϋπόθεση: όλες οι πράξεις να εκτελούνται χωρίς αναζήτηση οποιοδήποτε στοιχείου.

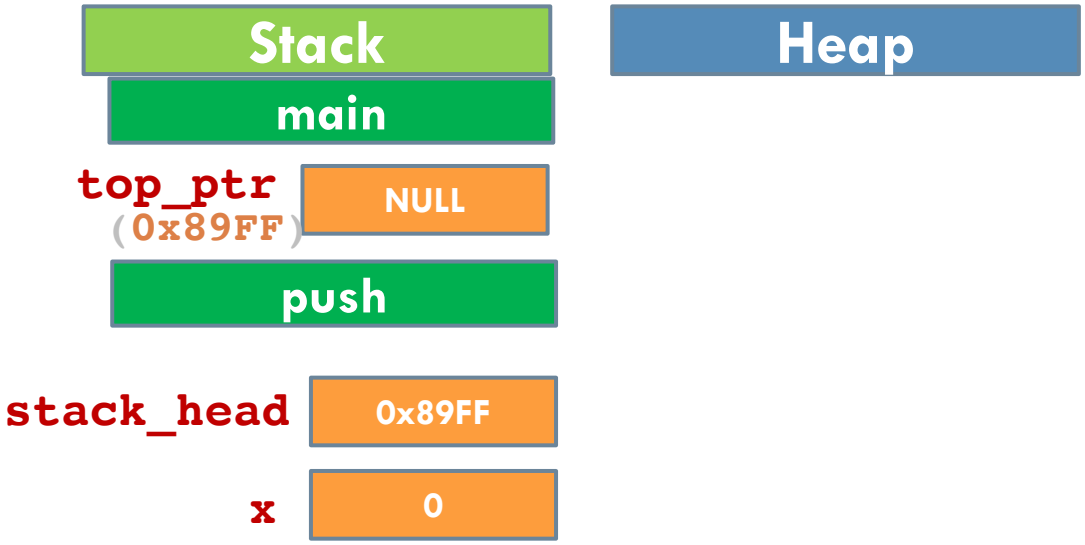
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

void push(StackNode **stack_head, int x );

int main()
{
    StackNode *top_ptr=NULL;

    return 0;
}

void push(StackNode **stack_head, int x )
{
    /**/
}
```



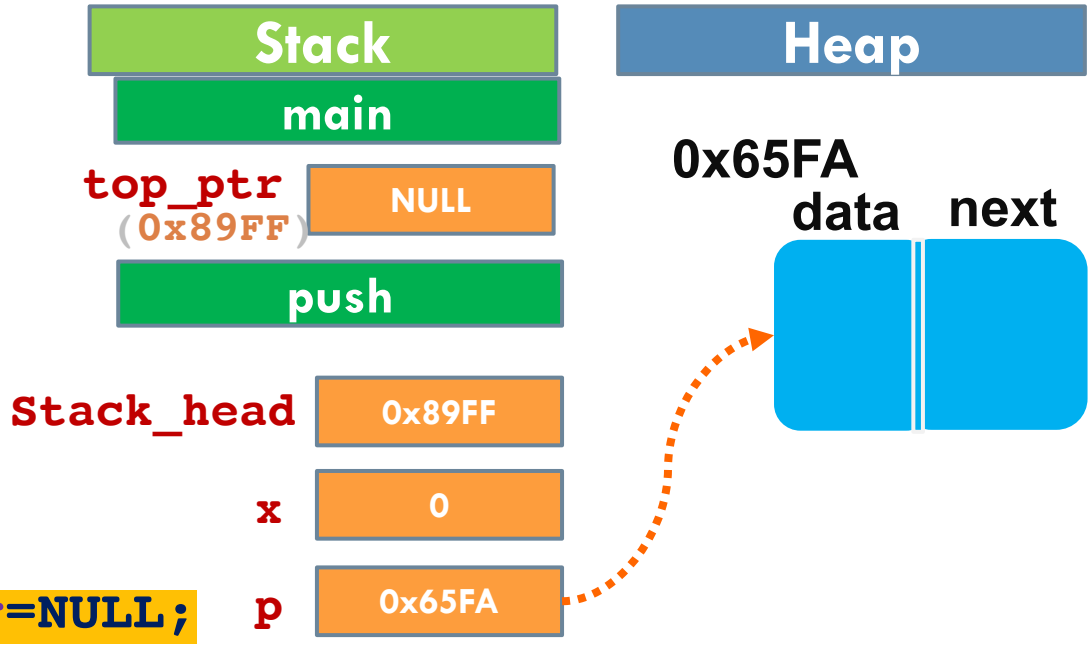
Στοίβα – Προσθήκη Κόμβου στη Στοίβα

```
void push(StackNode **stack_head, int x )
    {
    StackNode *p=malloc( sizeof ( StackNode ) );
        if (p!= NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
        }
    }
```

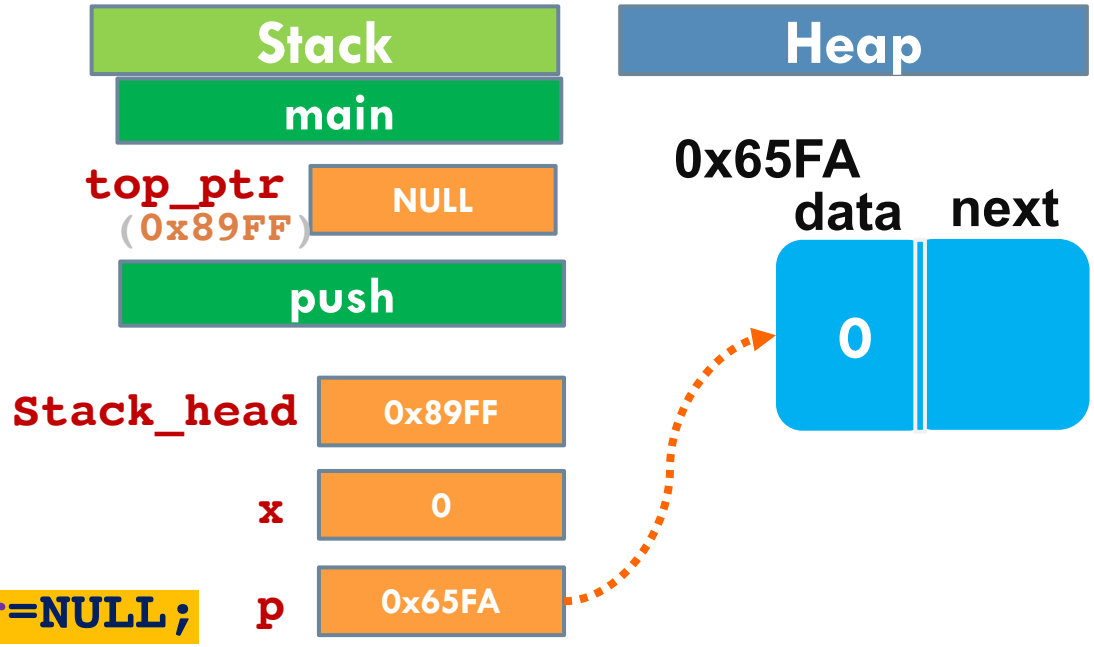
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
    StackNode *top_ptr=NULL;
    push(&top_ptr,0);
    return 0;
}

void push(StackNode **stack_head, int x )
{
    StackNode *p=malloc( sizeof ( StackNode ) );
    if ( p != NULL ) {
        p->data = x;
        p->next = *stack_head;
        *stack_head = p;
    }
```



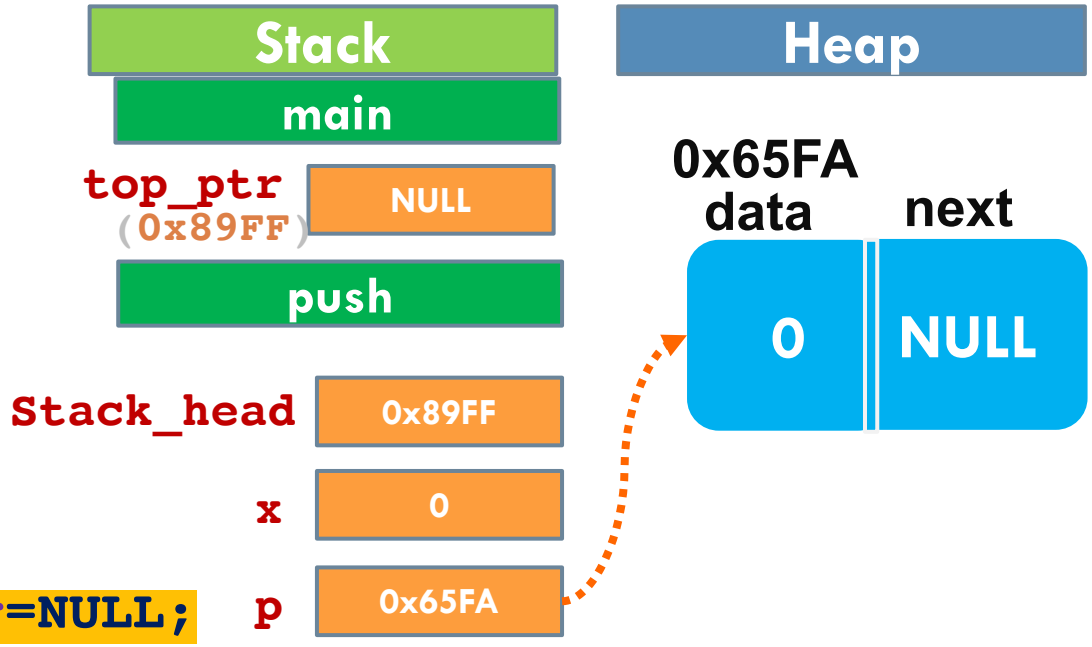
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
     if ( p != NULL ) {
          p->data = x;
          p->next = *stack_head;
          *stack_head = p;
     }
```



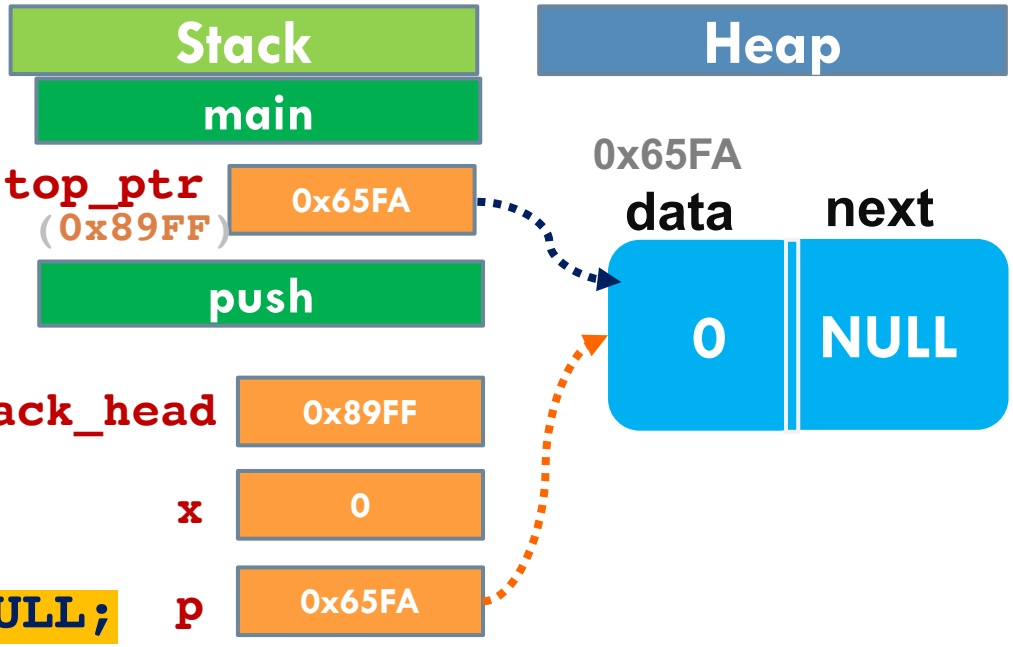
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
    StackNode *top_ptr=NULL;
    push(&top_ptr,0);
    return 0;
}

void push(StackNode **stack_head, int x )
{
    StackNode *p=malloc( sizeof ( StackNode ) );
    if ( p != NULL ) {
        p->data = x;
        p->next = *stack_head;
        *stack_head = p;
    }
```



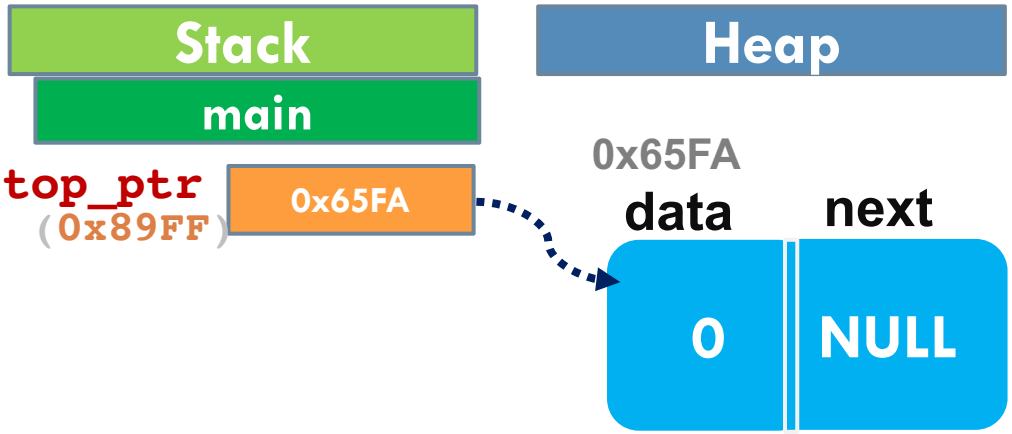
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
    StackNode *top_ptr=NULL;
    push(&top_ptr,0);
    return 0;
}

void push(StackNode **stack_head, int x )
{
    StackNode *p=malloc( sizeof ( StackNode ) );
    if ( p != NULL ) {
        p->data = x;
        p->next = *stack_head;
        *stack_head = p;
    }
```



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

int main()
{
    StackNode *top_ptr=NULL;
    push(&top_ptr,0);
    return 0;
}

void push(StackNode **stack_head, int x )
{
    StackNode *p=malloc( sizeof ( StackNode ) );
    if ( p != NULL ) {
        p->data = x;
        p->next = *stack_head;
        *stack_head = p;
    }
```

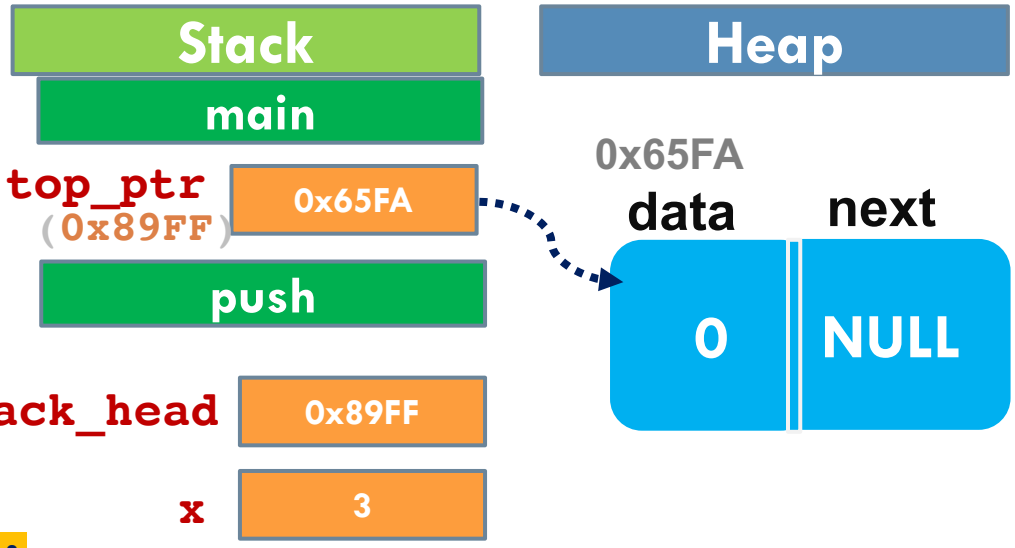


```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
    StackNode *p=malloc( sizeof ( StackNode ) );
     if ( p != NULL ) {
           p->data = x;
           p->next = *stack_head;
           *stack_head = p;
     }
}
```

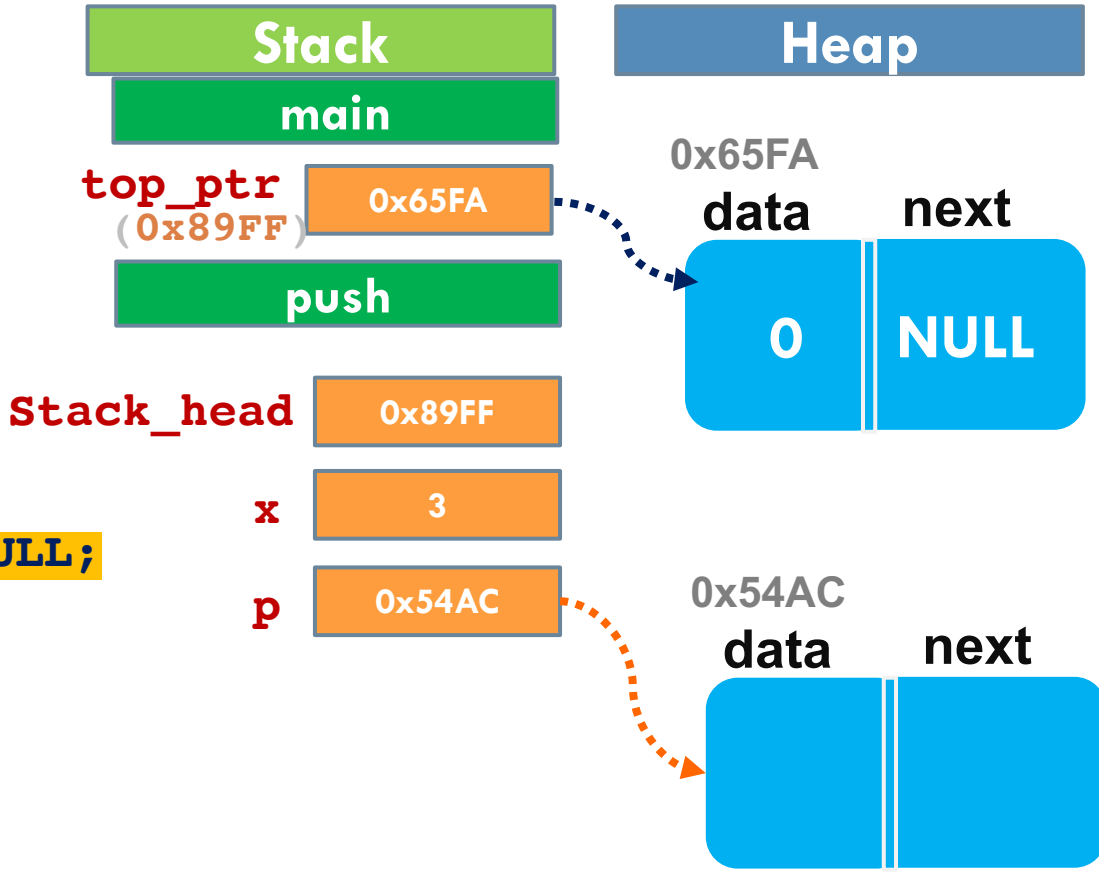


Θέλουμε να προσθέσουμε ένα δεύτερο κόμβο στη στοίβα

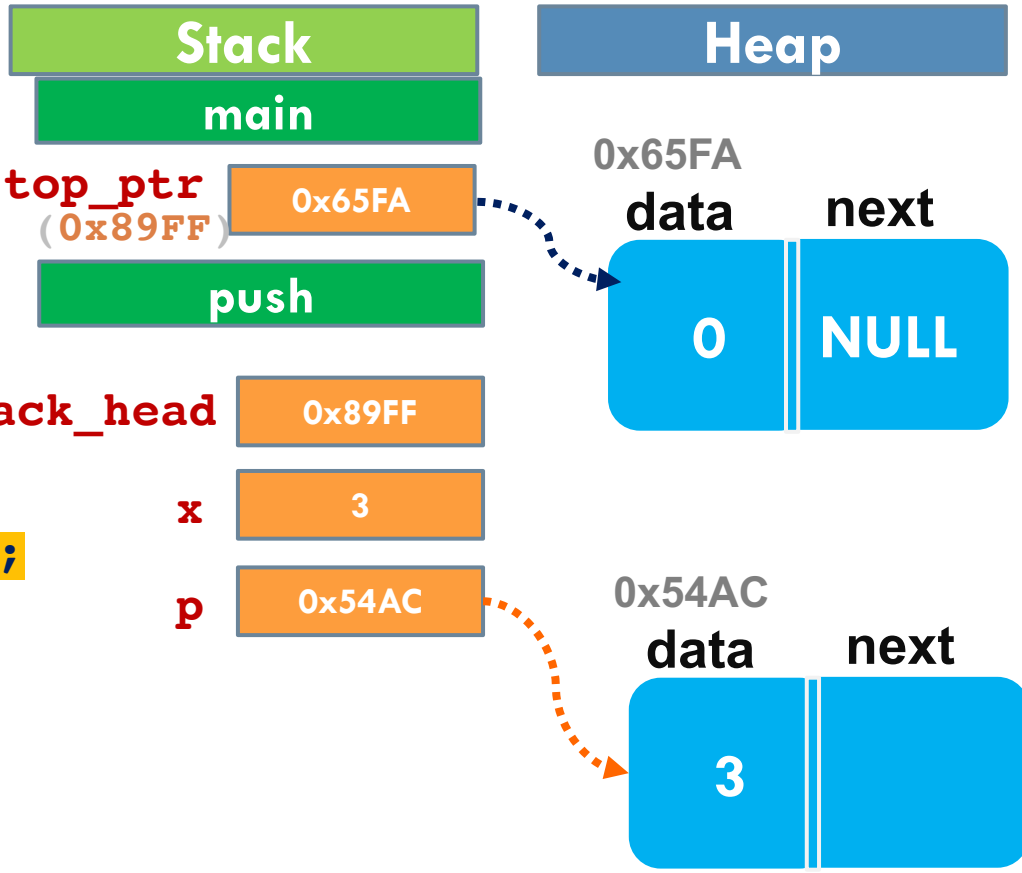
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
      if ( p != NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
      }
}
```



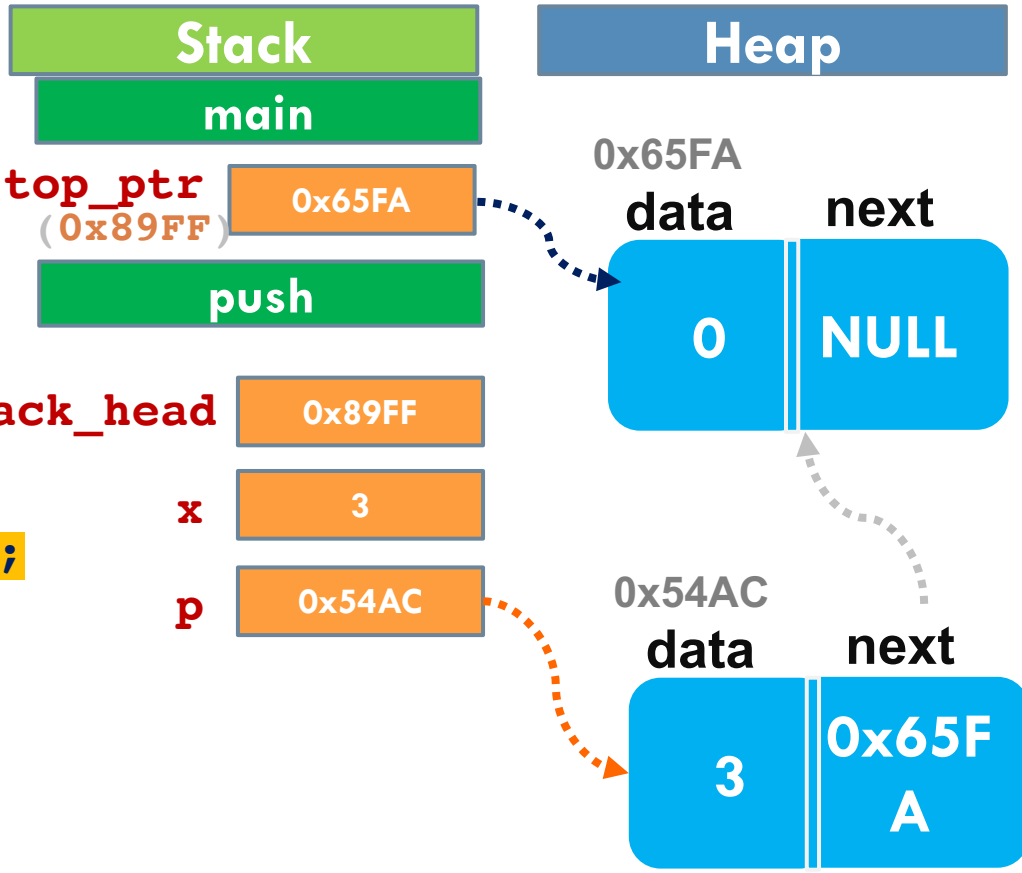
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
      if ( p != NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
      }
}
```



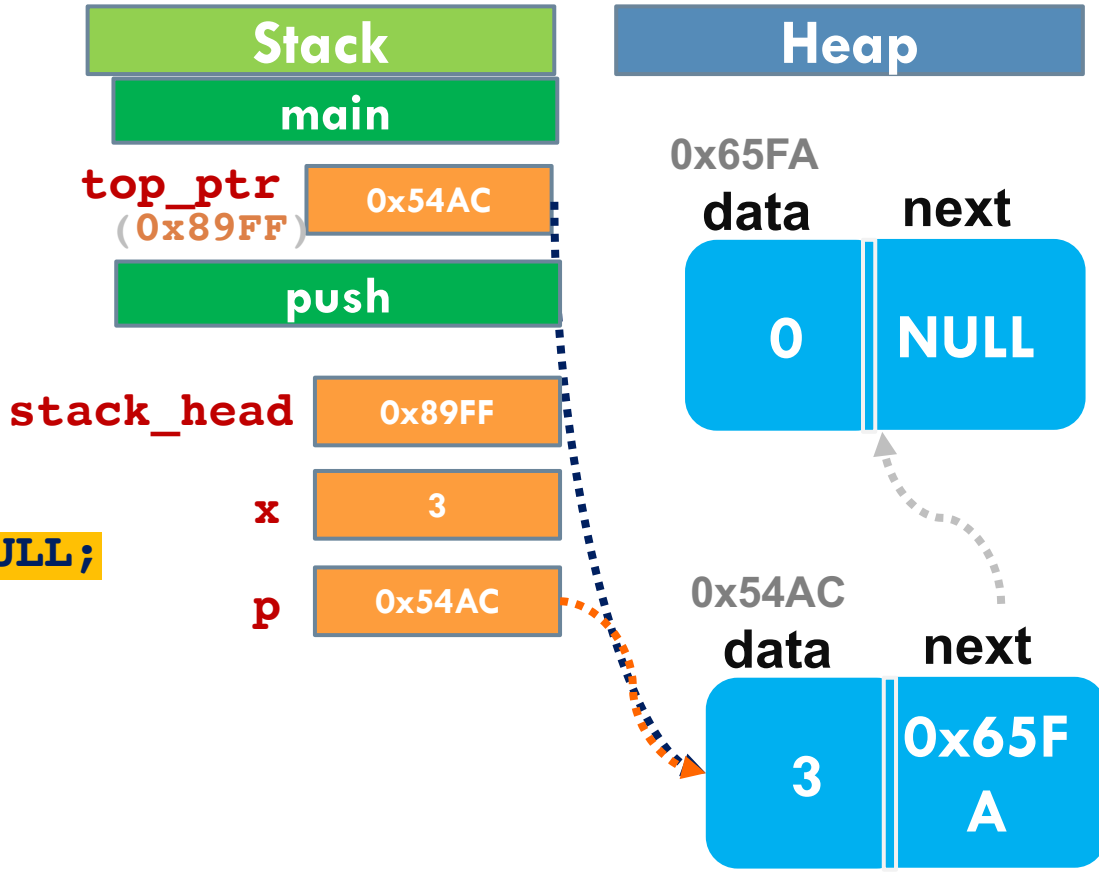
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
      if ( p != NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
      }
}
```



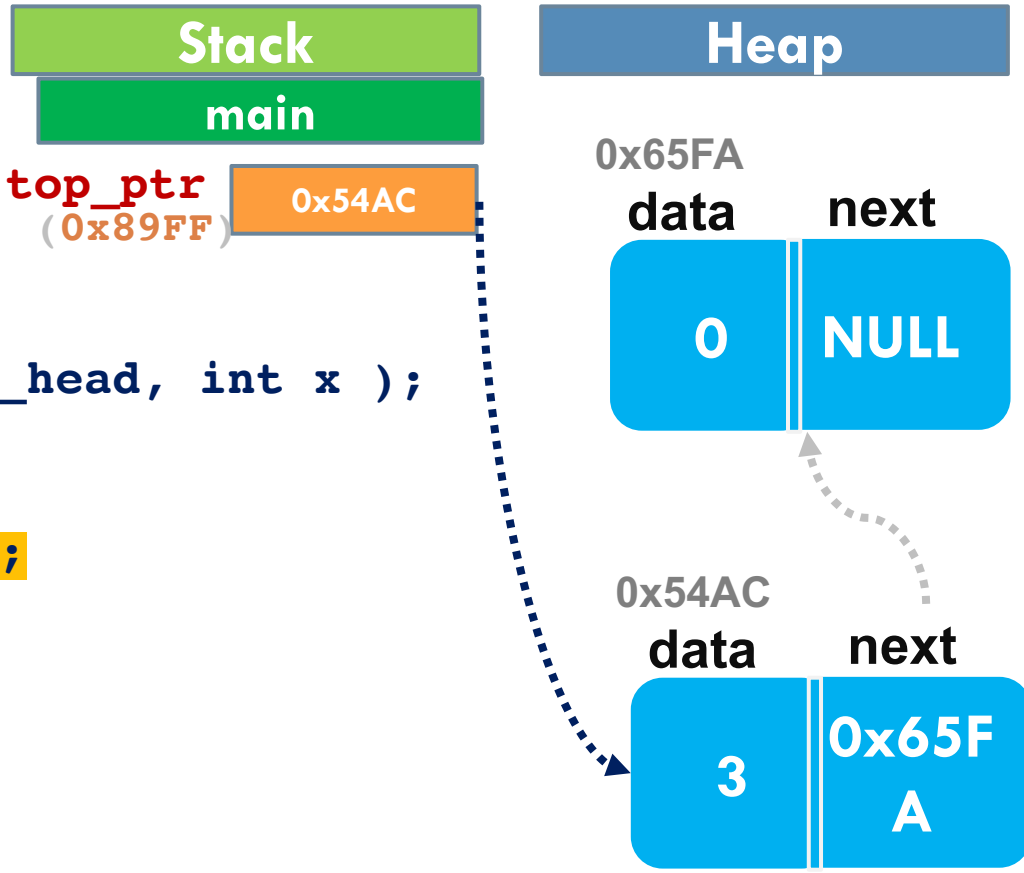
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
      if ( p != NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
      }
}
```



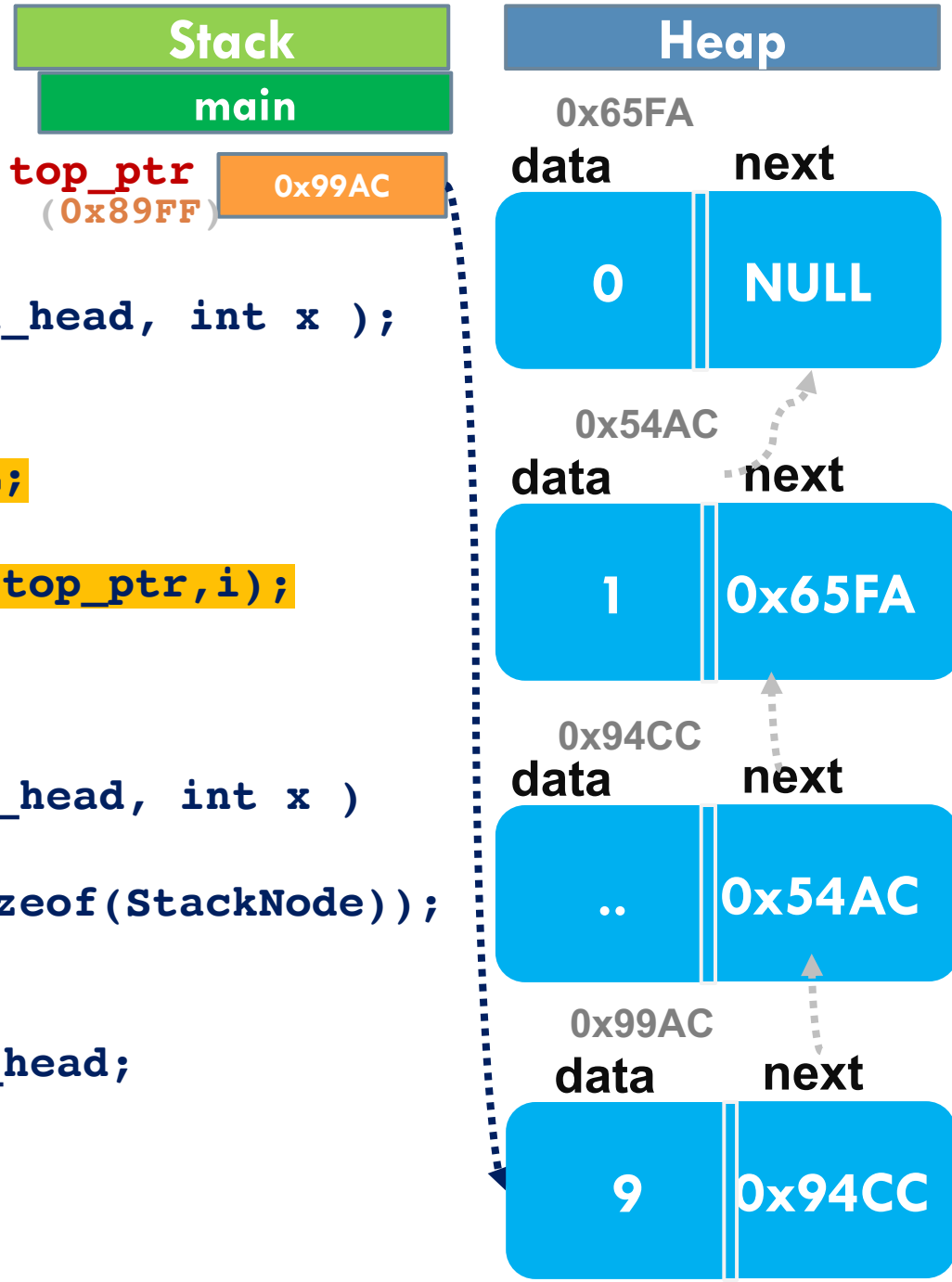
```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;

int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
      if ( p != NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
      }
}
```



```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;
void push(StackNode **stack_head, int x );
int main()
{
     StackNode *top_ptr=NULL;
     push(&top_ptr,0);
     push(&top_ptr,3);
     return 0;
}

void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof ( StackNode ) );
      if ( p != NULL ) {
            p->data = x;
            p->next = *stack_head;
            *stack_head = p;
      }
}
```



```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    int data;
    struct node *next;
} StackNode ;
void push(StackNode **stack_head, int x );
int main()
{
     StackNode *top_ptr=NULL;
     int i;
     for(i=0;i<10;i++)push(&top_ptr,i);

     return 0;
}
void push(StackNode **stack_head, int x )
{
     StackNode *p=malloc( sizeof(StackNode));
     if ( p != NULL ) {
          p->data = x;
          p->next = *stack_head;
          *stack_head = p;
     }
}
```

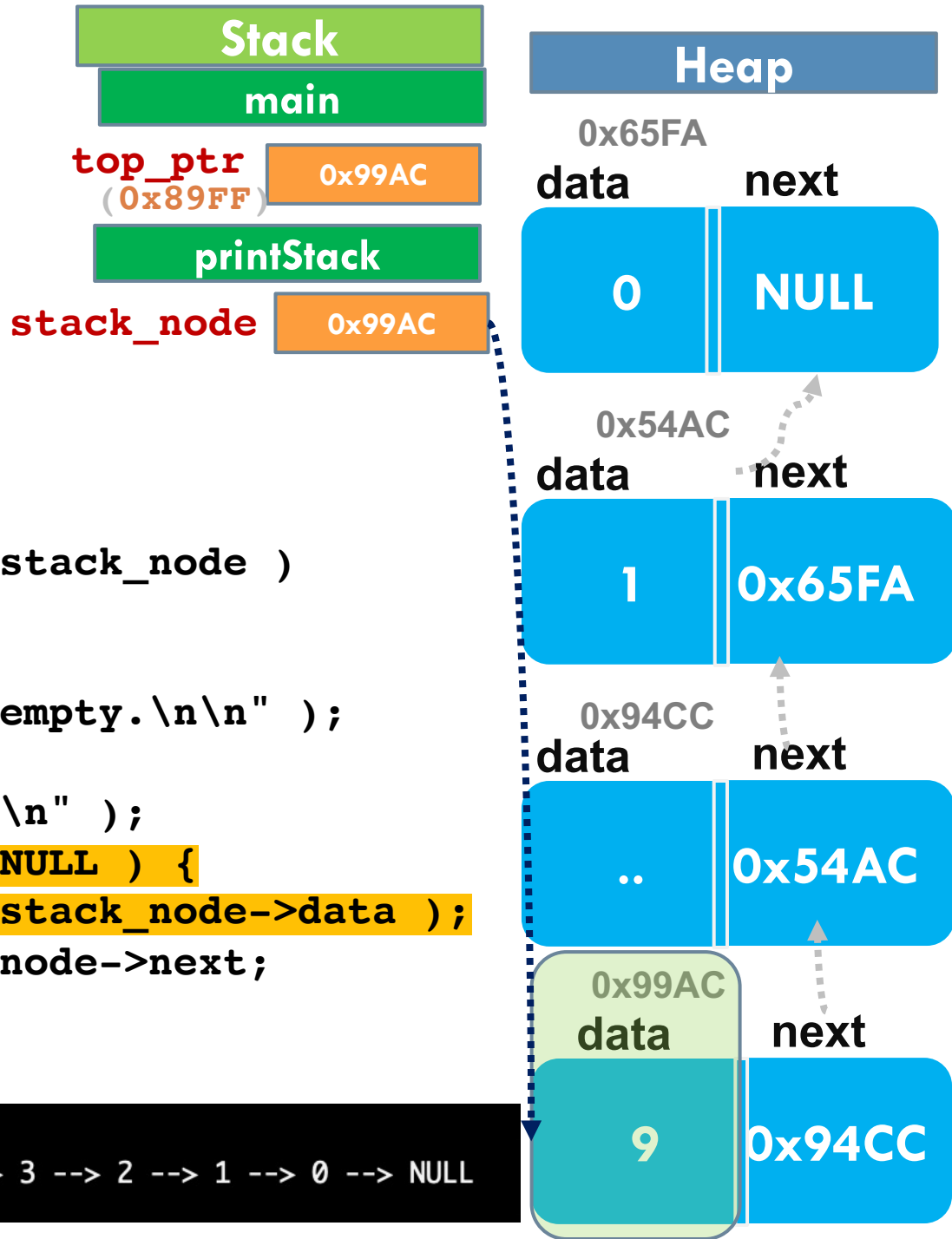


Στοίβα – Εκτύπωση Στοίβας

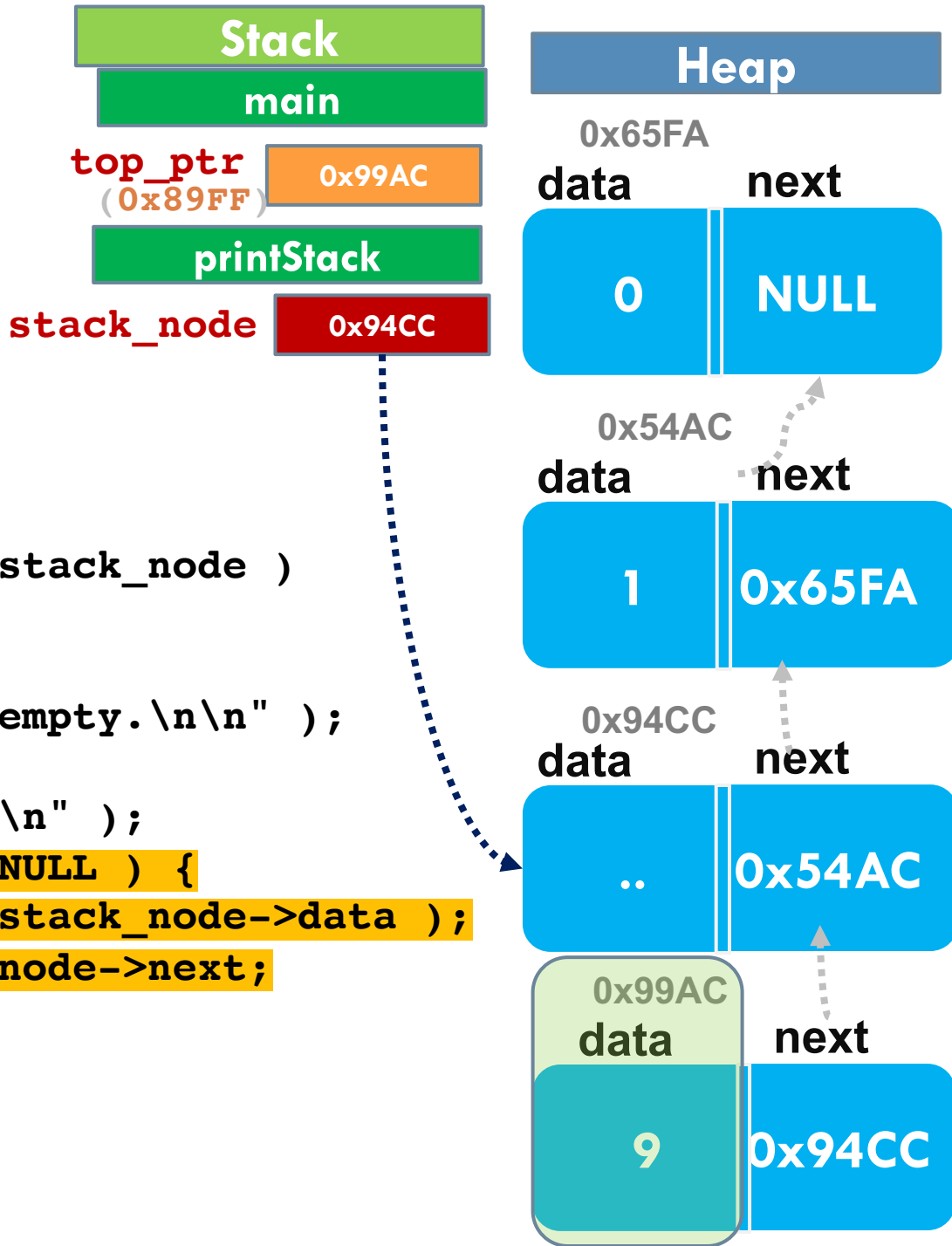
```
void printStack( StackNode *stack_node )
{
   if (stack_node == NULL )
      printf( "The stack is empty.\n\n" );
   else {
      printf( "The stack is:\n" );
      while ( stack_node != NULL ) {
         printf( "%d --> ", stack_node->data );
         stack_node = stack_node->next;
      }
      printf( "NULL\n\n" );
   }
}
```

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

     return 0;
}
void printStack( StackNode *stack_node )
{
    if (stack_node == NULL )
       printf( "The stack is empty.\n\n" );
    else {
       printf( "The stack is:\n" );
       while ( stack_node != NULL ) {
          printf( "%d --> ", stack_node->data );
          stack_node = stack_node->next;
       }
       printf( "NULL\n\n" );
    }
}
```

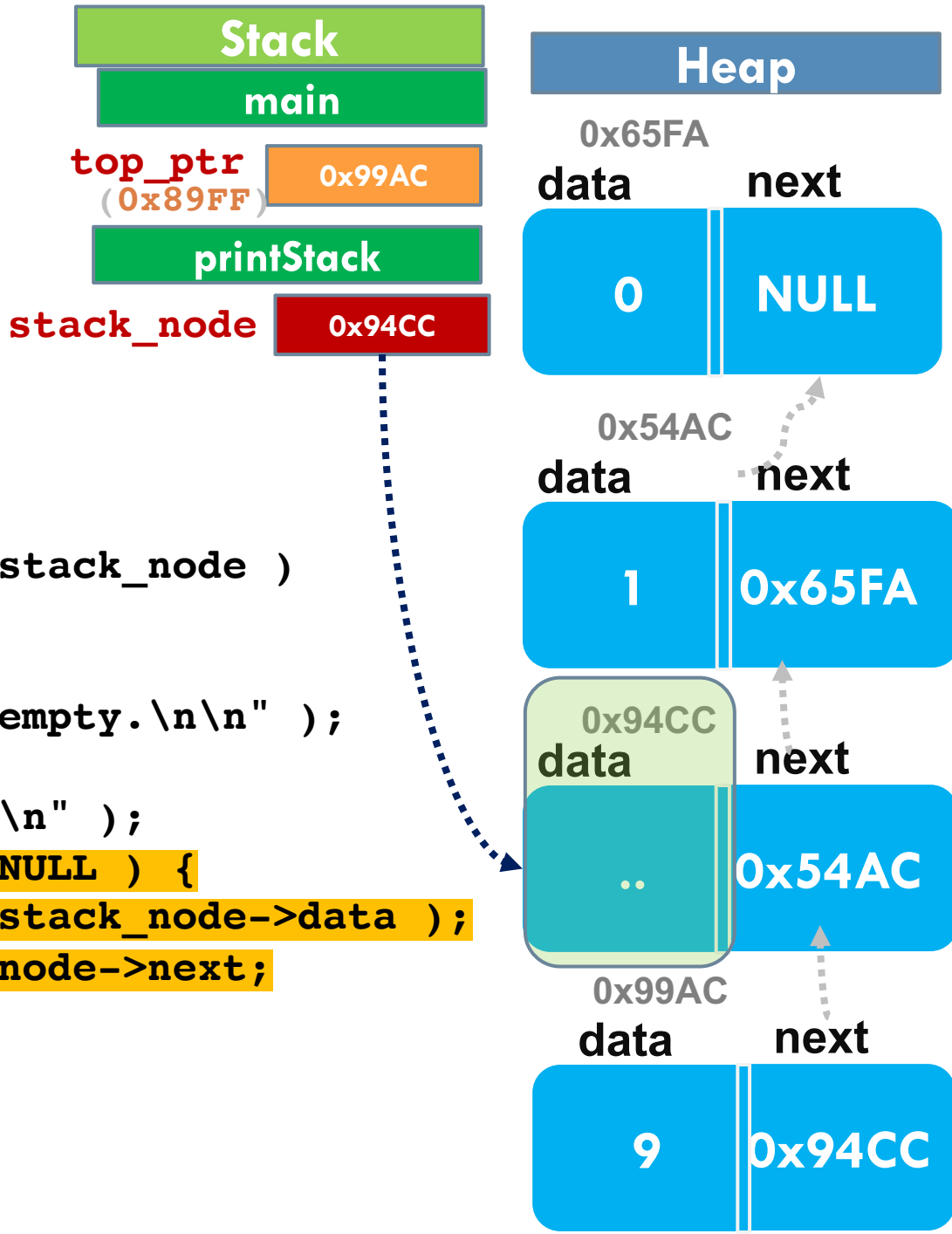
```
The stack is:
9 --> 8 --> 7 --> 6 --> 5 --> 4 --> 3 --> 2 --> 1 --> 0 --> NULL
```



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

     return 0;
}
void printStack( StackNode *stack_node )
{
    if (stack_node == NULL )
       printf( "The stack is empty.\n\n" );
    else {
       printf( "The stack is:\n" );
       while ( stack_node != NULL ) {
          printf( "%d --> ", stack_node->data );
          stack_node = stack_node->next;
       }
       printf( "NULL\n\n" );
    }
}
```



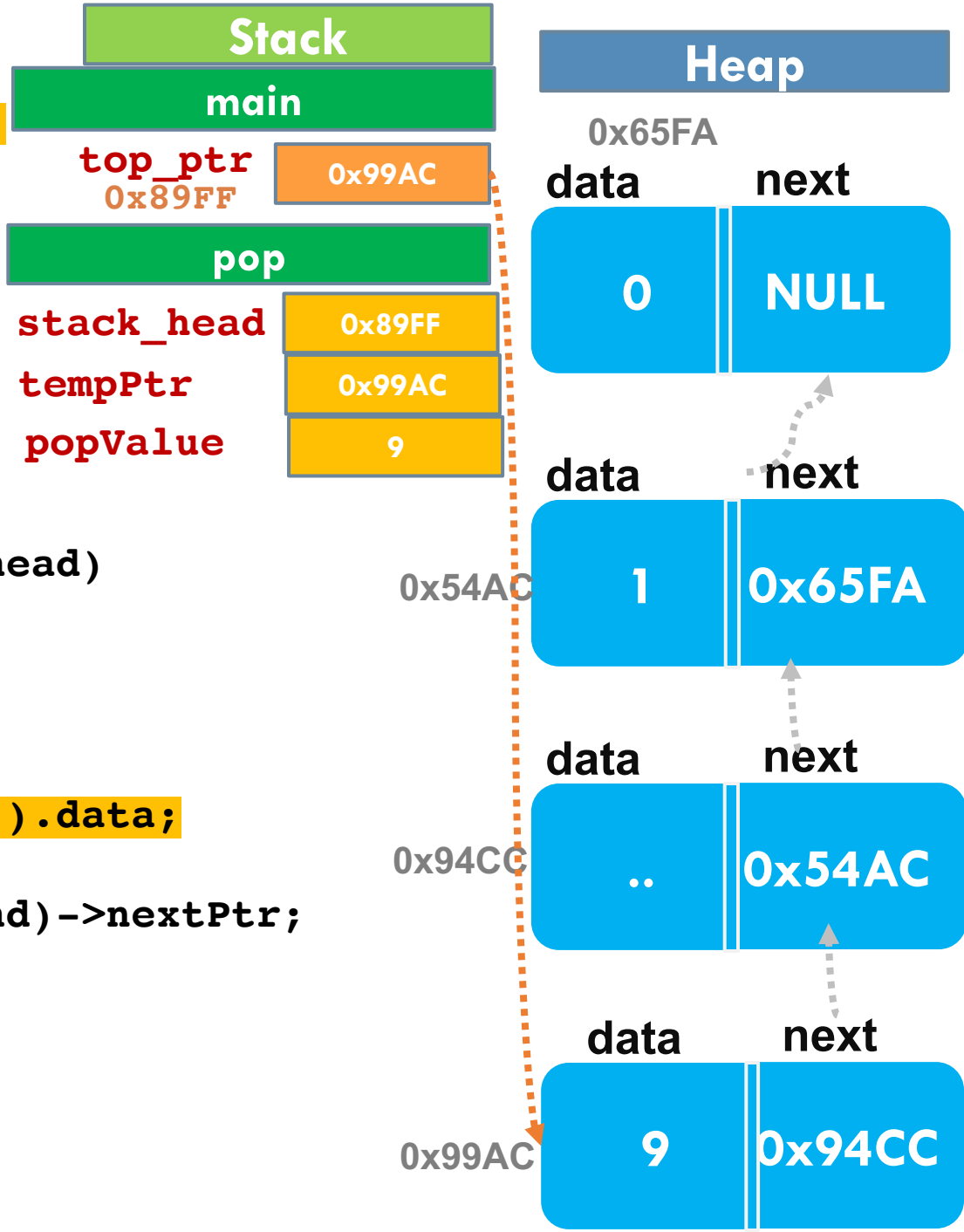
```c
int main()
{
     StackNode *top_ptr=NULL;
      int i;
      for(i=0;i<10;i++){
         push(&top_ptr,i);
      }
     printStack(top_ptr);

     return 0;
}
void printStack( StackNode *stack_node )
{
    if (stack_node == NULL )
       printf( "The stack is empty.\n\n" );
    else {
       printf( "The stack is:\n" );
       while ( stack_node != NULL ) {
          printf( "%d --> ", stack_node->data );
          stack_node = stack_node->next;
       }
       printf( "NULL\n\n" );
    }
}
```



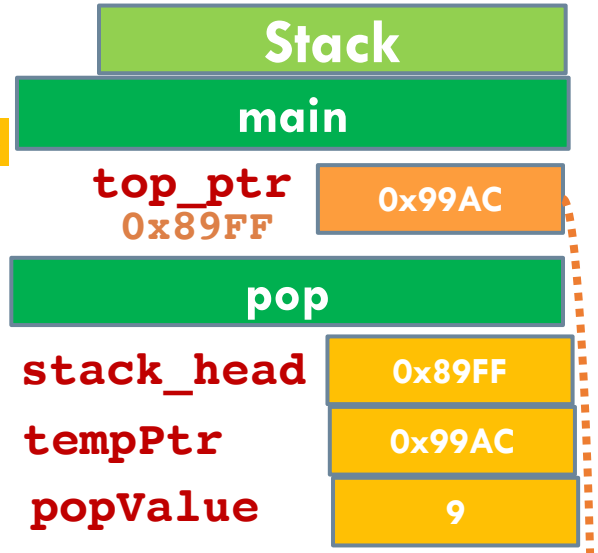
Στοίβα – Διαγραφή κόμβου κορυφής

```
int pop( StackNode **stack_head)
{
   StackNode *tempPtr;
   tempPtr = *stack_head;

   int popValue;
   popValue = ( **stack_head ).data;

   *stack_head = ( *stack_head)->next;
   free( tempPtr );
   return popValue;
}
```

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


int pop( StackNode **stack_head)
{
    StackNode *tempPtr;
    tempPtr = *stack_head;
    int popValue;
    popValue = (**stack_head ).data;

    *stack_head = (*stack_head)->nextPtr;
    free( tempPtr );
    return popValue;
}
```

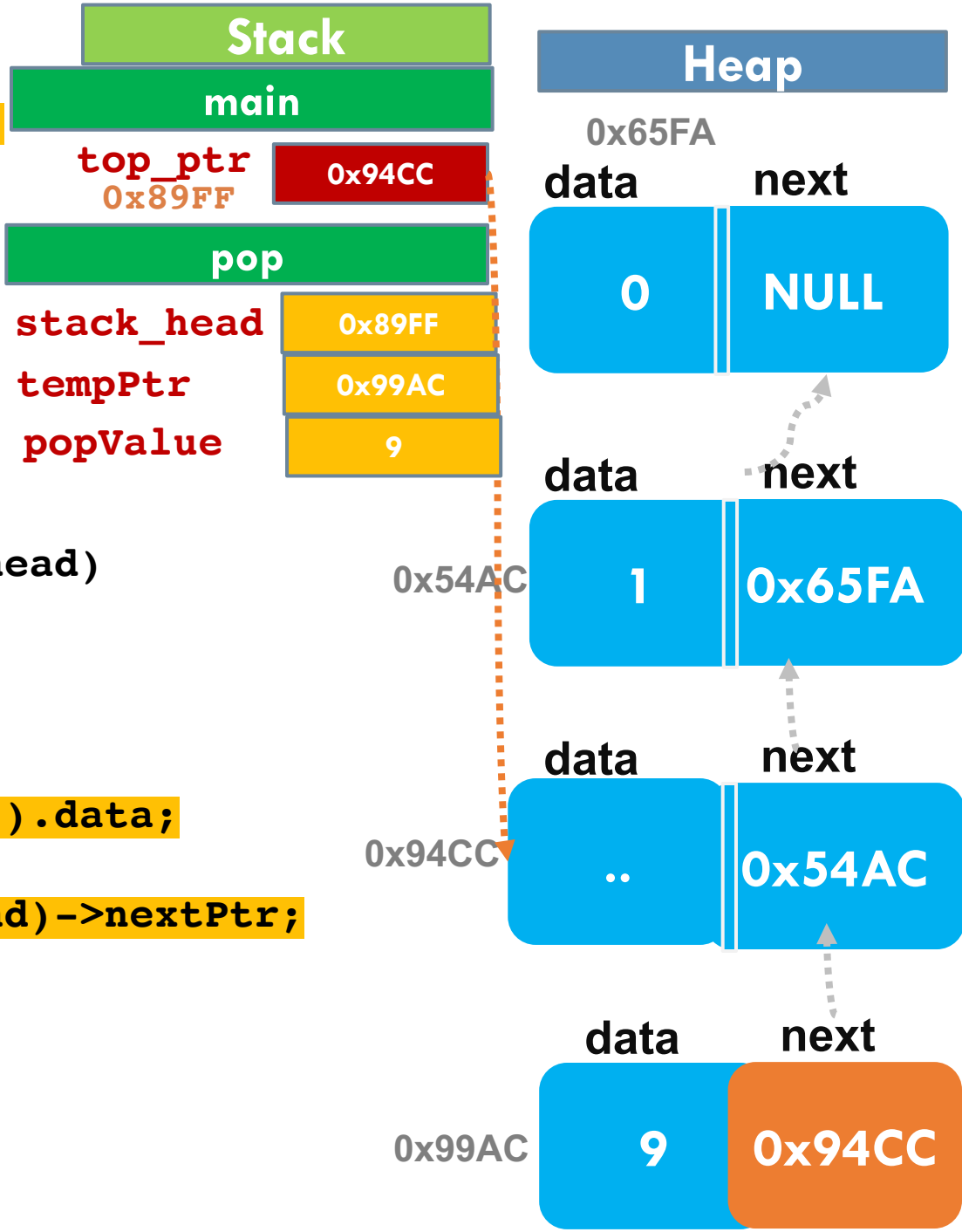


```
int main()
{
     StackNode *top_ptr=NULL;
      int i;
      for(i=0;i<10;i++)
          push(&top_ptr,i);
     pop(&top_ptr);
     return 0;
}
```

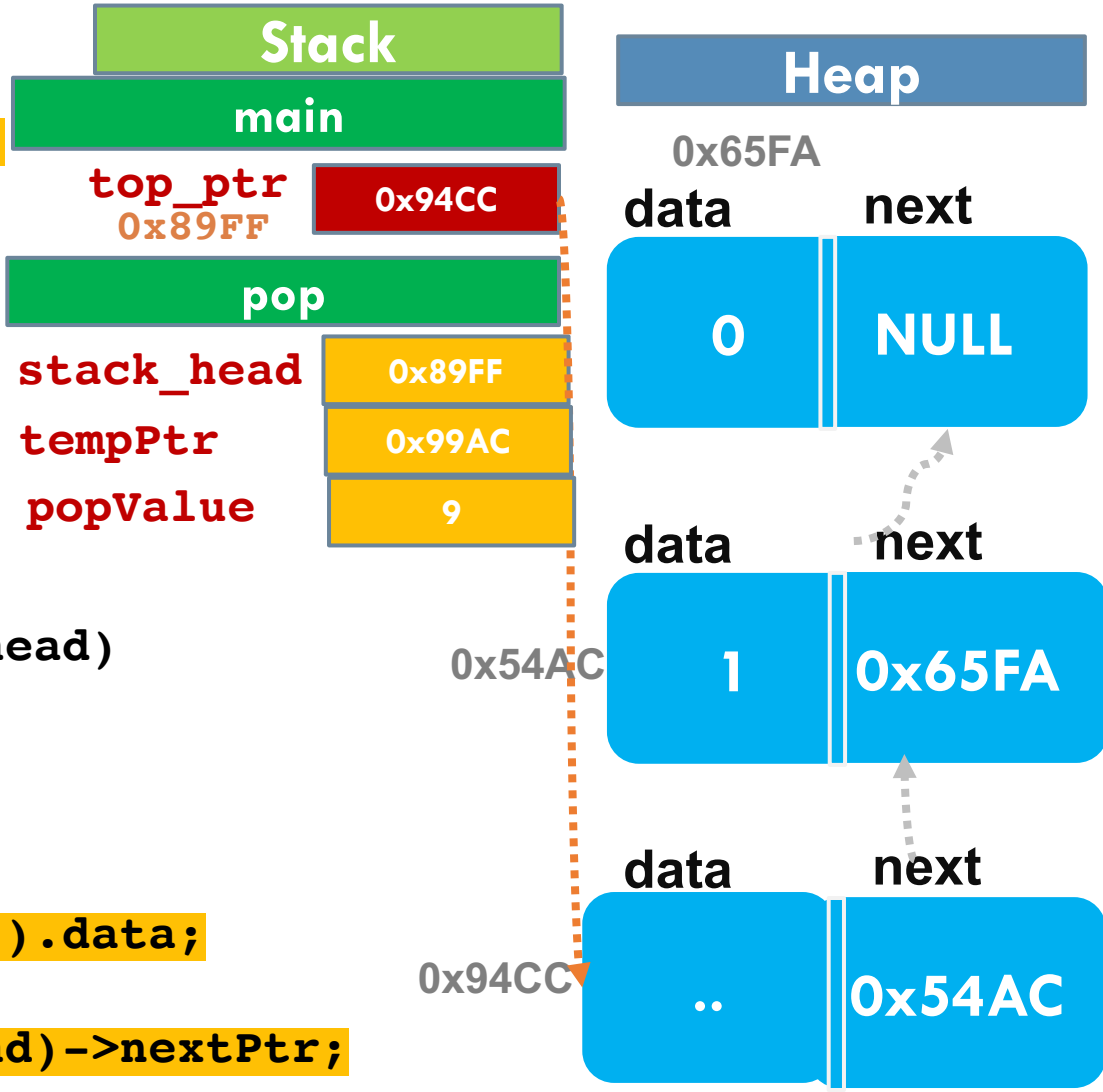



```
int pop( StackNode **stack_head)
{
    StackNode *tempPtr;
    int popValue;
    popValue = (**stack_head ).data;

    *stack_head = (*stack_head)->nextPtr;

    tempPtr = *stack_head;
    free( tempPtr );
    return popValue;
}
```

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


int pop( StackNode **stack_head)
{
    StackNode *tempPtr;
    tempPtr = *stack_head;
    int popValue;
    popValue = (**stack_head ).data;

    *stack_head = (*stack_head)->nextPtr;

    free( tempPtr );
    return popValue;
}
```



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


int pop( StackNode **stack_head)
{
    StackNode *tempPtr;
    tempPtr = *stack_head;
    int popValue;
    popValue = (**stack_head ).data;

    *stack_head = (*stack_head)->nextPtr;

    free( tempPtr );
    return popValue;
}
```



Ακόμη ένα Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
struct stackNode {
   int data;
   struct stackNode *nextPtr;
};
typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;

void push( StackNodePtr *, int );
int pop( StackNodePtr * );
int isEmpty( StackNodePtr );
void printStack( StackNodePtr );
void instructions( void );
```

Ορισμός τύπων
&
Δήλωση προτύπων συναρτήσεων

```c
int main() {
   StackNodePtr stackPtr = NULL;  /* points to stack top */
   int choice, value;

   instructions();
   printf( "? " );
   scanf( "%d", &choice );

   while ( choice != 3 ) {

      switch ( choice ) {
         case 1:      /* push value onto stack */
            printf( "Enter an integer: " );
            scanf( "%d", &value );
            push( &stackPtr, value );
            printStack( stackPtr );
            break;
         case 2:      /* pop value off stack */
            if ( !isEmpty( stackPtr ) )
               printf( "The popped value is %d.\n",
               pop( &stackPtr ) );
               printStack( stackPtr );
            break;
         default:
            printf( "Invalid choice.\n\n" );
            instructions();
            break;
      }

      printf( "? " );
      scanf( "%d", &choice );
   }

   printf( "End of run.\n" );
   return 0;
}
```

Υλοποίηση Βασικής Συνάρτησης

```c
/* Print the instructions */
void instructions( void )
{
   printf( "Enter choice:\n"
           "1 to push a value on the stack\n"
           "2 to pop a value off the stack\n"
           "3 to end program\n" );
}
/* Insert a node at the stack top */
void push( StackNodePtr *topPtr, int info )
{
   StackNodePtr newPtr;
   newPtr = malloc( sizeof ( StackNode ) );
   if ( newPtr != NULL ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   }
   else
      printf( "%d not inserted. No memory available.\n", info );
}
```

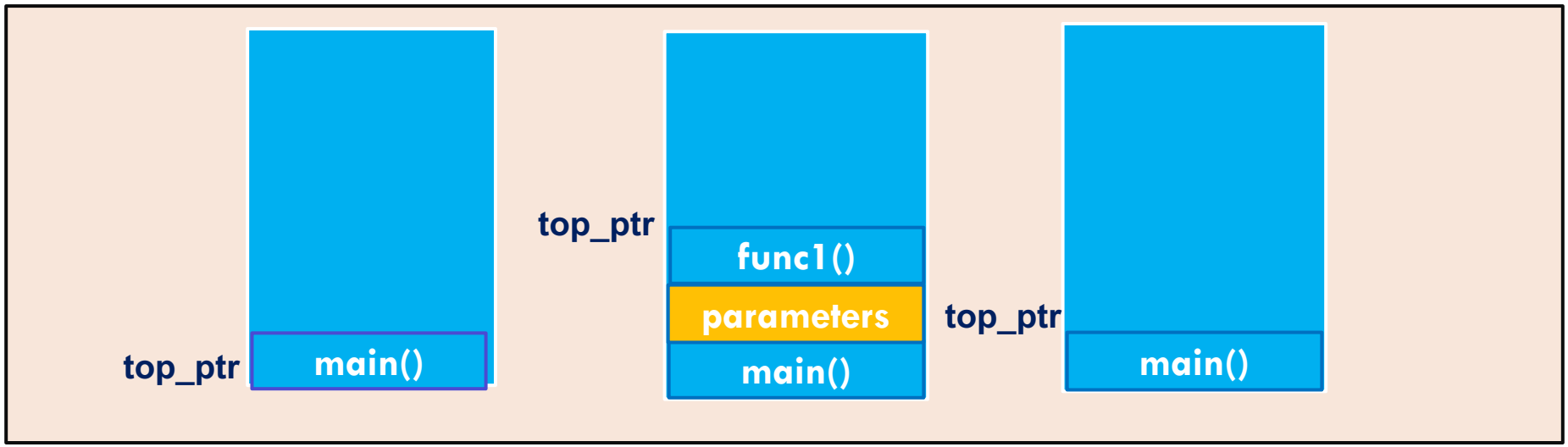
Υλοποίηση Λοιπών Συναρτήσεων

```c
/* Print the stack */
void printStack( StackNodePtr currentPtr )
{
   if ( currentPtr == NULL )
      printf( "The stack is empty.\n\n" );
   else {
      printf( "The stack is:\n" );

      while ( currentPtr != NULL ) {
         printf( "%d --> ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      }

      printf( "NULL\n\n" );
   }
}
```

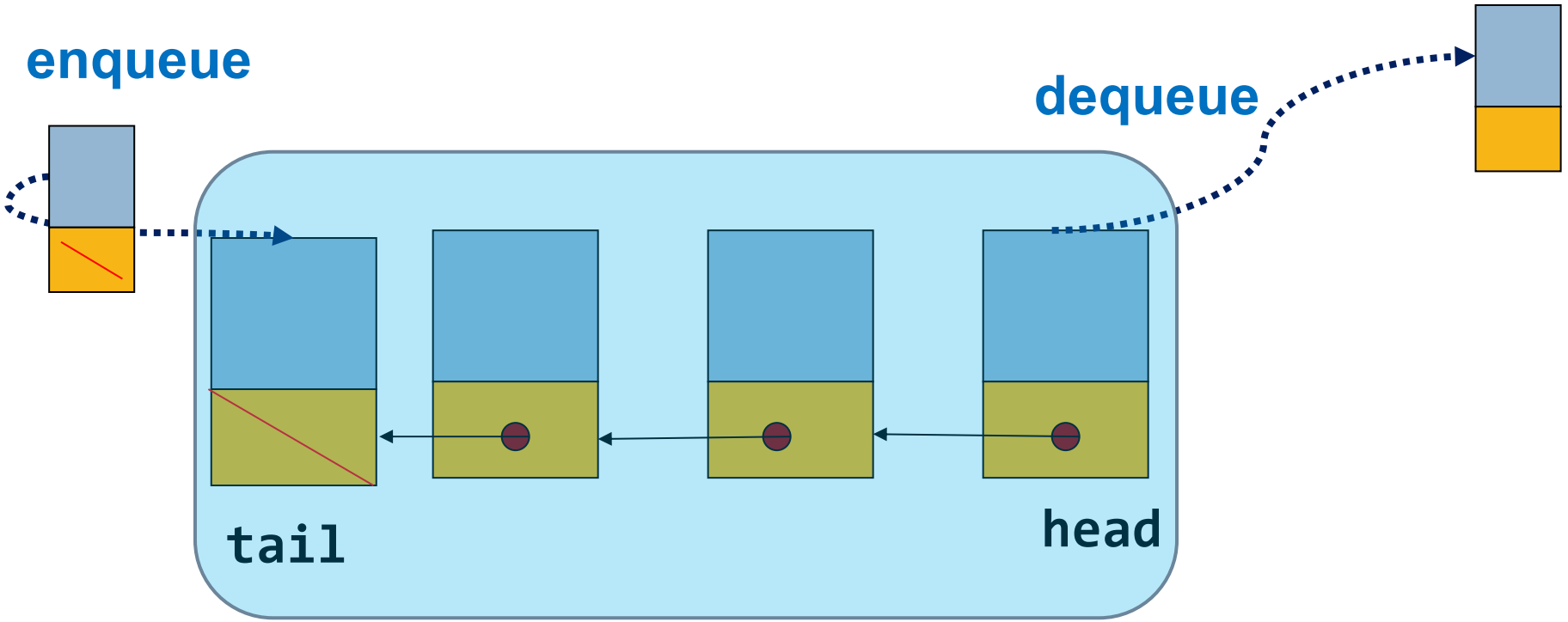
```c
/* Is the stack empty? */
int isEmpty( StackNodePtr topPtr )
{
   return topPtr == NULL;
}

/* Remove a node from the stack top */
int pop( StackNodePtr *topPtr )
{
   StackNodePtr tempPtr;
   int popValue;

   tempPtr = *topPtr;
   popValue = ( *topPtr )->data;
   *topPtr = ( *topPtr )->nextPtr;
   free( tempPtr );
   return popValue;
}
```

- Οι στοίβες βρίσκουν μεγάλη χρήση στην Πληροφορική για δημιουργία άλλων δομών και σε βασικό λογισμικό.

- Κλασικό παράδειγμα αφορά την κλήση υποπρογραμμάτων (function calls) και αναδρομικών διαδικασιών.
  - Σε κάθε κλήση οποιασδήποτε συνάρτησης ένα σύνολο από λέξεις (stack frame) φυλάσσεται σε μια στοίβα, από όπου μπορεί να ανασυρθεί.
  - Όταν μια συνάρτηση καλεί μια άλλη συνάρτηση οι παράμετροι της συνάρτησης, η διεύθυνση επιστροφής και οι τοπικές μεταβλητές της καλούσας συνάρτησης φυλάσσονται μέσα στη στοίβα του προγράμματος.
  - Έτσι, όταν η κληθείσα συνάρτηση τερματίσει, το περιβάλλον την καλούσας συνάρτησης ανασύρεται από τη στοίβα για να συνεχιστεί κανονικά η εκτέλεσή της.



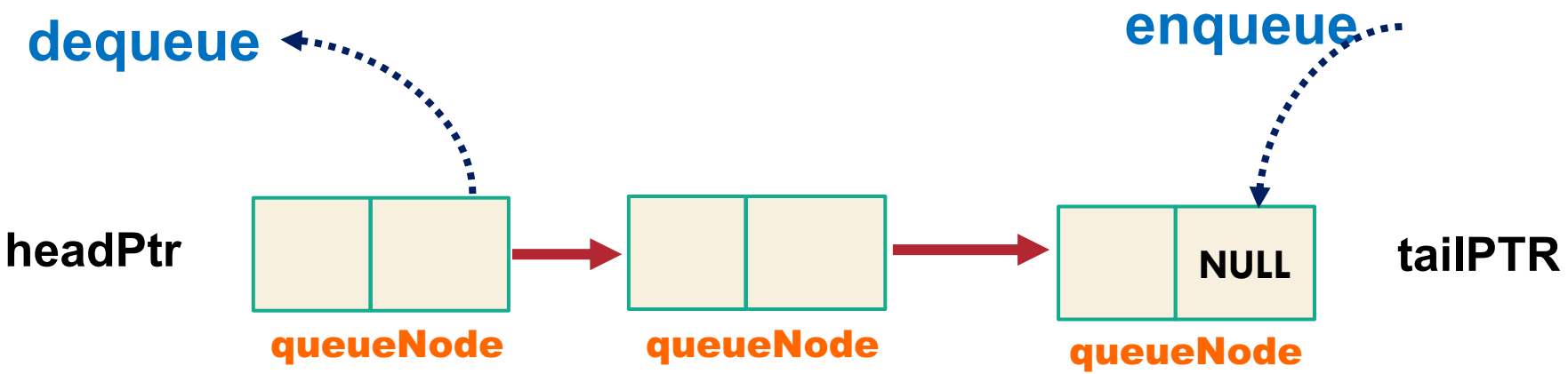
Ουρές - queues



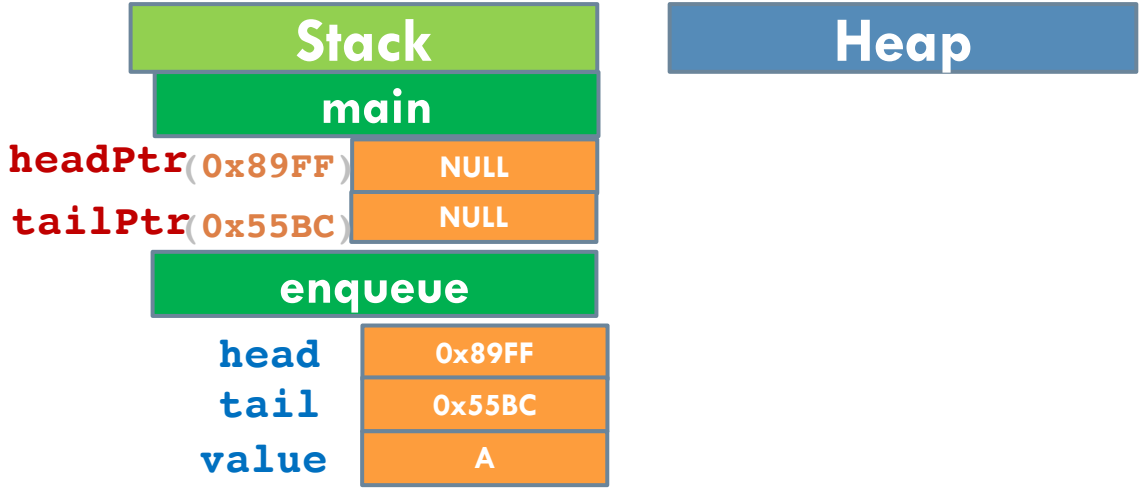
- Λειτουργίες enqueue/dequeue
- First-In First-Out (FIFO)

Queue –Υλοποίηση με συνδεδεμένη λίστα

```
typedef struct node {
                char date;
                struct node *next;
} QueueNode;
```

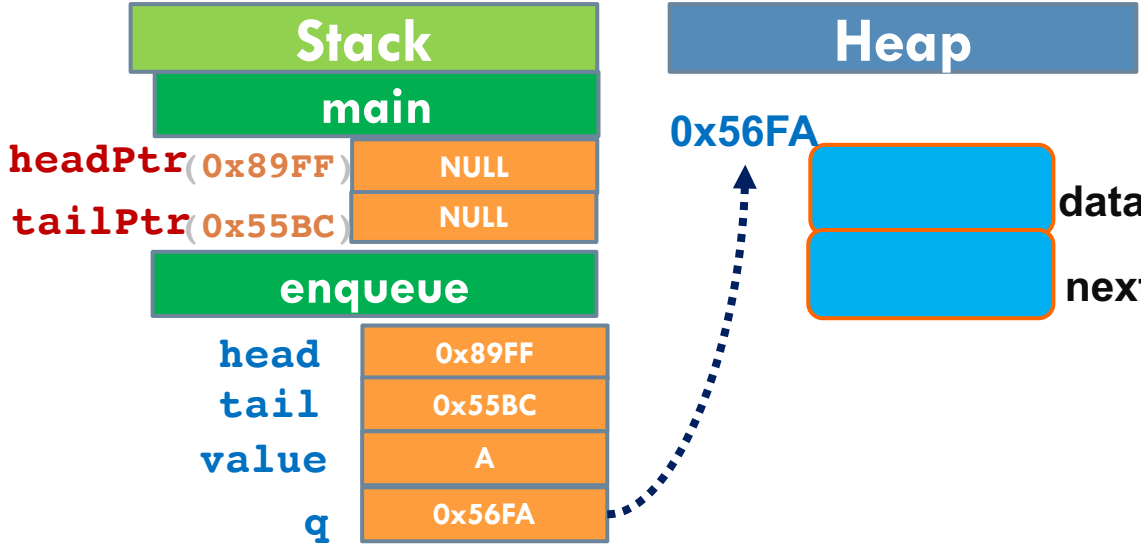


```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



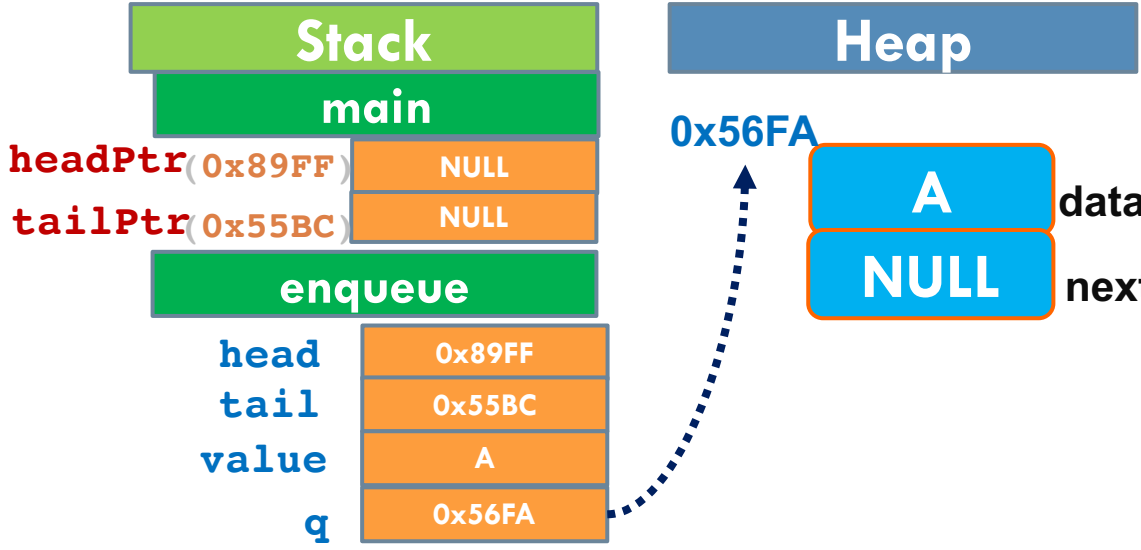

```c
void enqueue( QueueNode **, QueueNode **, char );
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
  if ( q != NULL ) {
     q->data = value;
     q->next = NULL;

     if ( *head==NULL )
        *head = q;
      else
        ( *tail )->next = q;

  *tail = q;
  }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



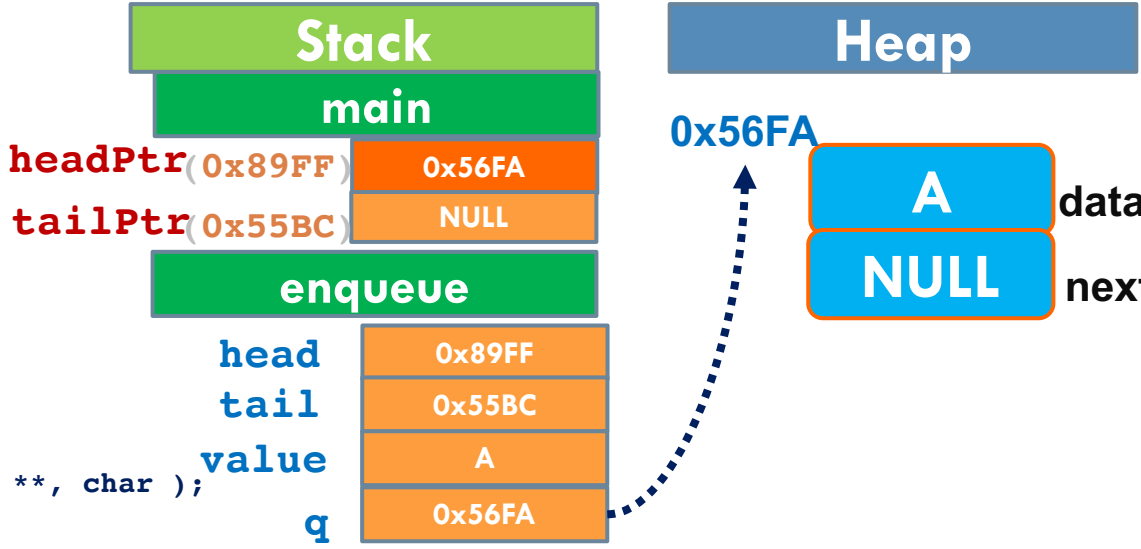

```c
void enqueue( QueueNode **, QueueNode **, char );
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
  if ( q != NULL ) {
     q->data = value;
     q->next = NULL;

     if ( *head==NULL )
        *head = q;
      else
        ( *tail )->next = q;

  *tail = q;
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```

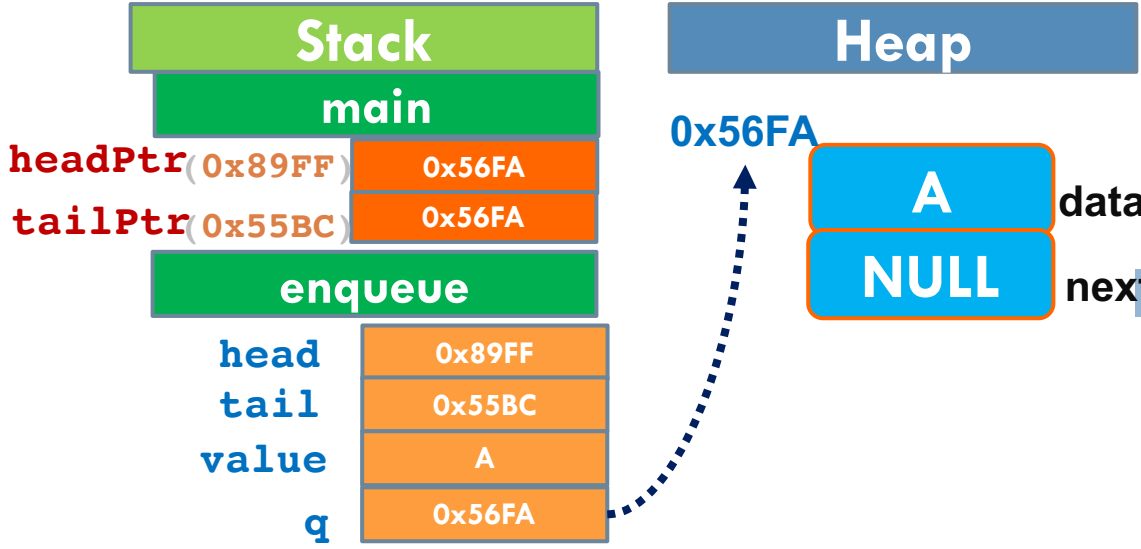



```c
void enqueue( QueueNode **, QueueNode **, char );
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
  if ( q != NULL ) {
      q->data = value;
      q->next = NULL;

      if ( *head==NULL )
         *head = q;
      else
         ( *tail )->next = q;

  *tail = q;
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;

void enqueue( QueueNode **, QueueNode **, char );
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
     return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
  if ( q != NULL ) {
       q->data = value;
       q->next = NULL;

       if ( *head==NULL )
          *head = q;
       else
          ( *tail )->next = q;

  *tail = q;
  }
}
```

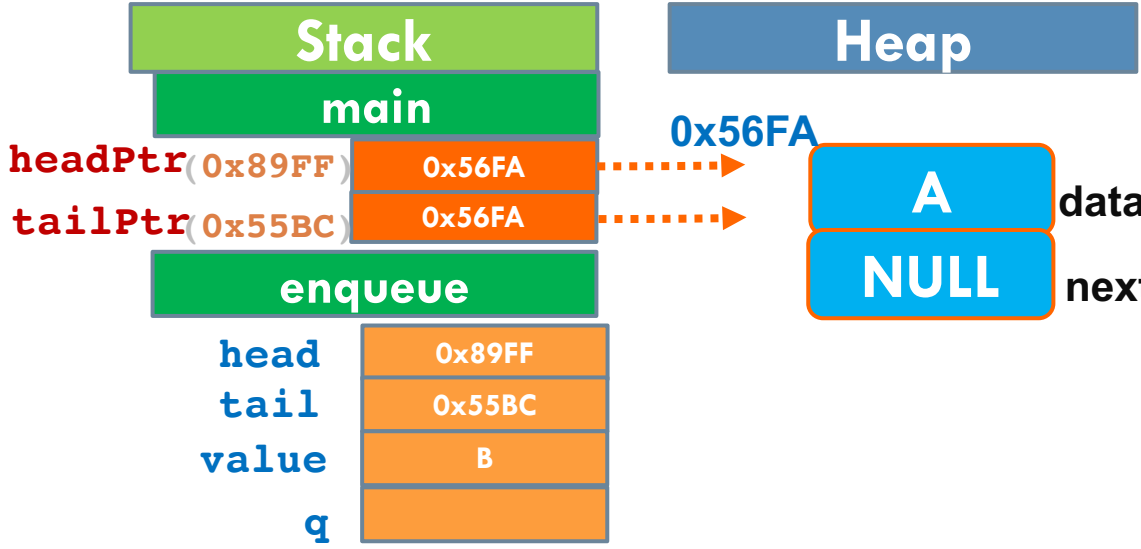


```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



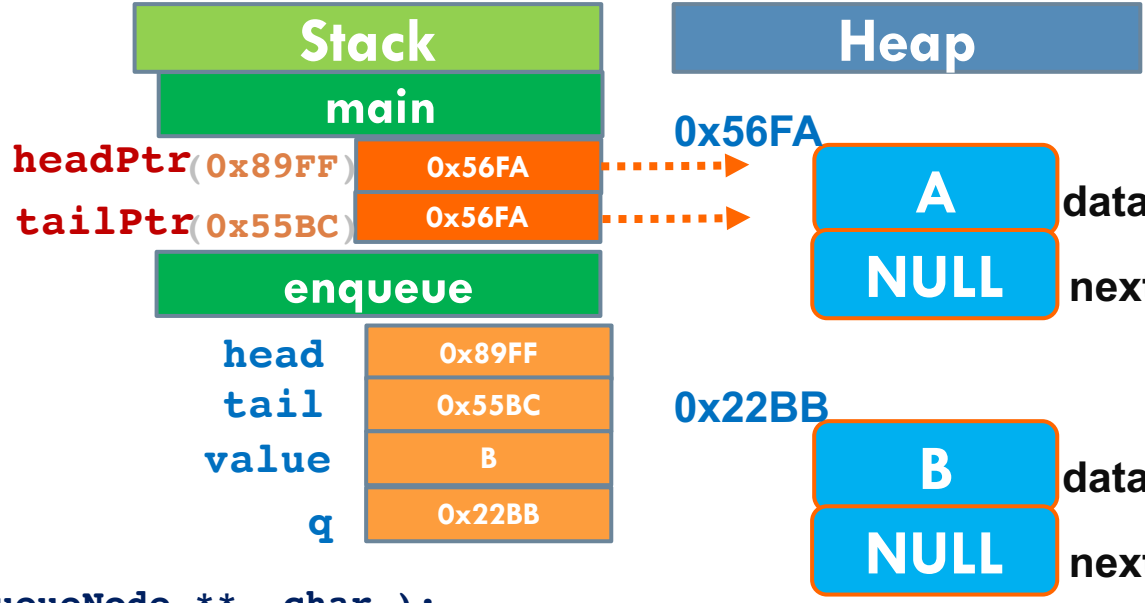

```c
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
       enqueue( &headPtr, &tailPtr, 'A');
     return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
  if ( q != NULL ) {
      q->data = value;
      q->next = NULL;

      if ( *head==NULL )
         *head = q;
       else
          ( *tail )->next = q;

  *tail = q;
  }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



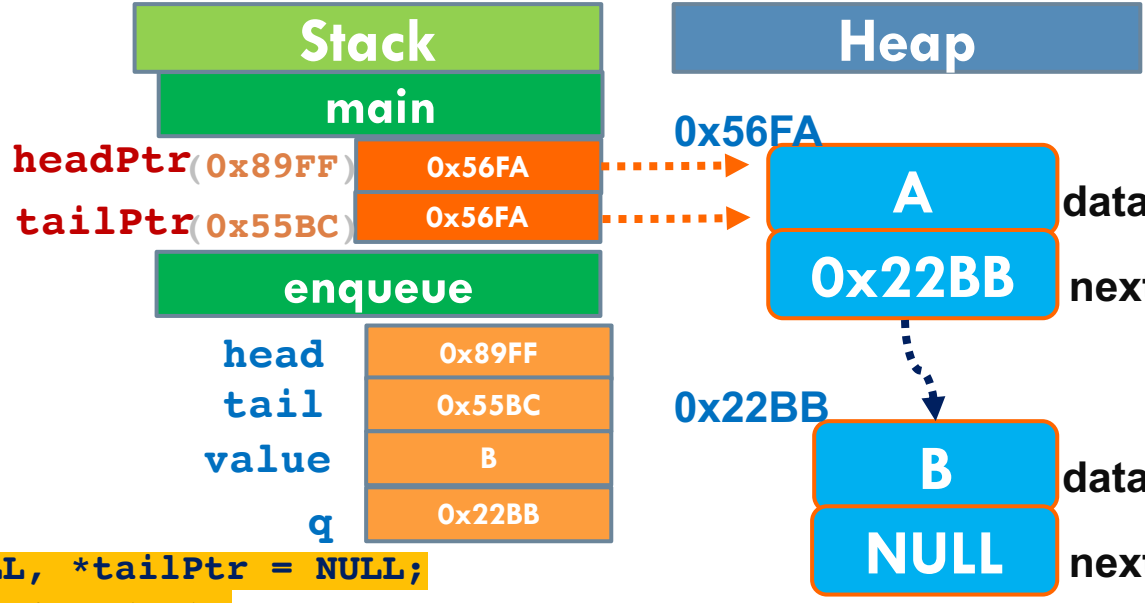

```c
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
  if ( q != NULL ) {
     q->data = value;
     q->next = NULL;

     if ( *head==NULL )
        *head = q;
      else
        ( *tail )->next = q;

  *tail = q;
  }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



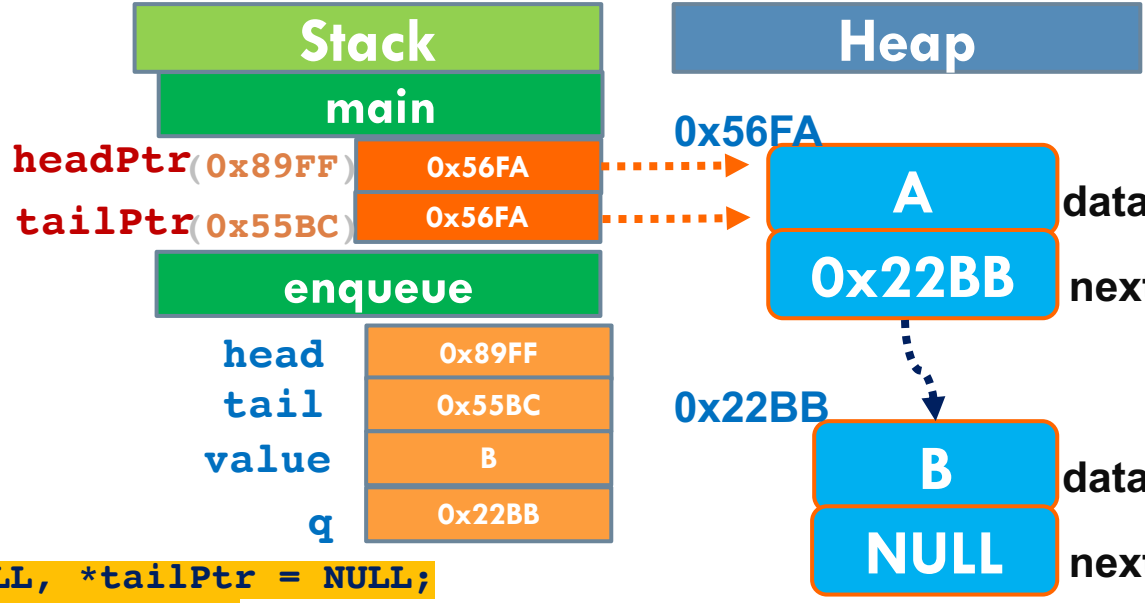

```c
void enqueue( QueueNode **, QueueNode **, char );
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
    if ( q != NULL ) {
       q->data = value;
       q->next = NULL;

      if ( *head==NULL )
         *head = q;
       else
         ( *tail )->next = q;
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```

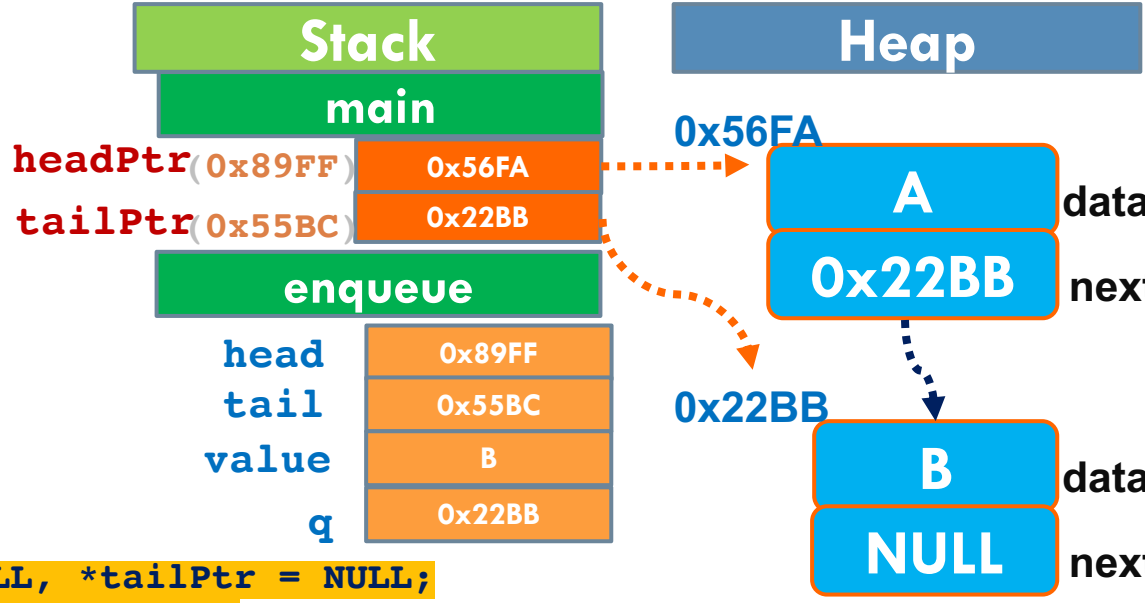



```c
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
       enqueue( &headPtr, &tailPtr, 'A');
       enqueue( &headPtr, &tailPtr, 'B');
     return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
    if ( q != NULL ) {
       q->data = value;
       q->next = NULL;

       if ( *head==NULL )
          *head = q;
       else
          ( *tail )->next = q;

  *tail = q;
  }
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;


int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
    if ( q != NULL ) {
       q->data = value;
       q->next = NULL;

       if ( *head==NULL )
          *head = q;
       else
          ( *tail )->next = q;

  *tail = q;
  }
}
```



```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;


int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
    return 0;
}
void enqueue( QueueNode **head, QueueNode **tail, char value){
  QueueNode *q = malloc( sizeof( QueueNode ) );
    if ( q != NULL ) {
       q->data = value;
       q->next = NULL;

       if ( *head==NULL )
          *head = q;
       else
          ( *tail )->next = q;

  *tail = q;
  }
}
```



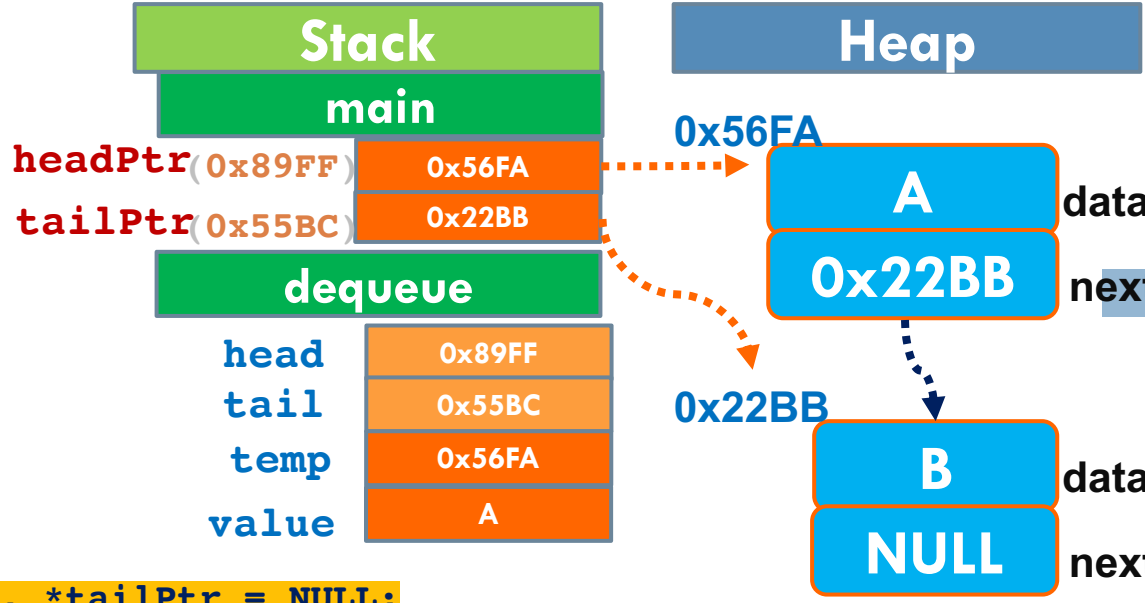
Ουρές – Εκτύπωση κόμβων

```
void printQueue( QueueNode *currentPtr )
{
   if ( currentPtr == NULL )
      printf( "Queue is empty.\n\n" );
   else {
      printf( "The queue is:\n" );

      while ( currentPtr != NULL ) {
         printf( "%c --> ", currentPtr->data );
         currentPtr = currentPtr->next;
      }
      printf( "NULL\n\n" );
   }
}
```

Ουρές – Dequeue

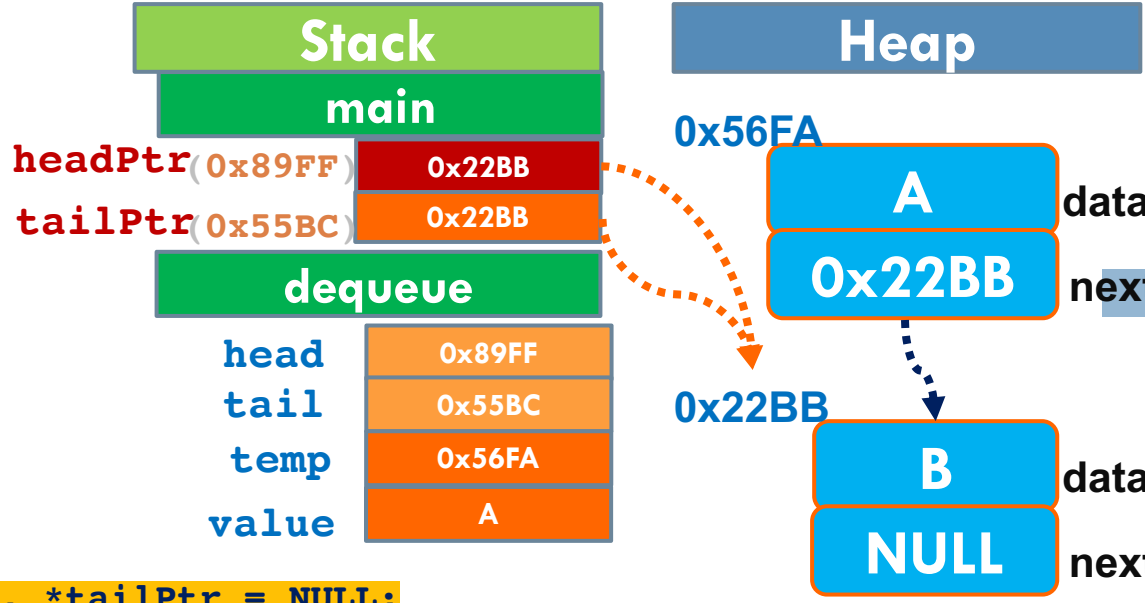
```
char dequeue( QueueNode **head, QueueNode **tail )
{
   char value;
   QueueNode *temp=*head;

   value = ( *head )->data;
   *head = ( *head )->next;

   if ( *head == NULL )
      *tail = NULL;

   free( temp );
   return value;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



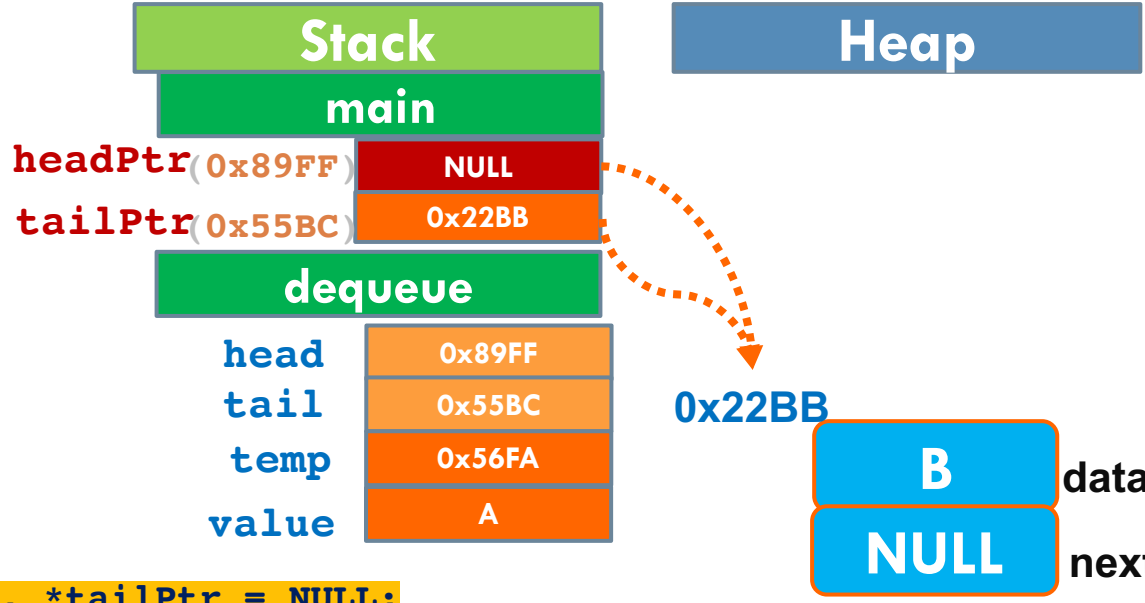

```c
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
      dequeue(&headPtr, &tailPtr);
    return 0;
}
char dequeue( QueueNode **head, QueueNode **tail ){
    char value;
    QueueNode *temp=*head;

    value = ( *head )->data;
    *head = ( *head )->next;

    if ( *head == NULL )
       *tail = NULL;

    free( temp );
    return value;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



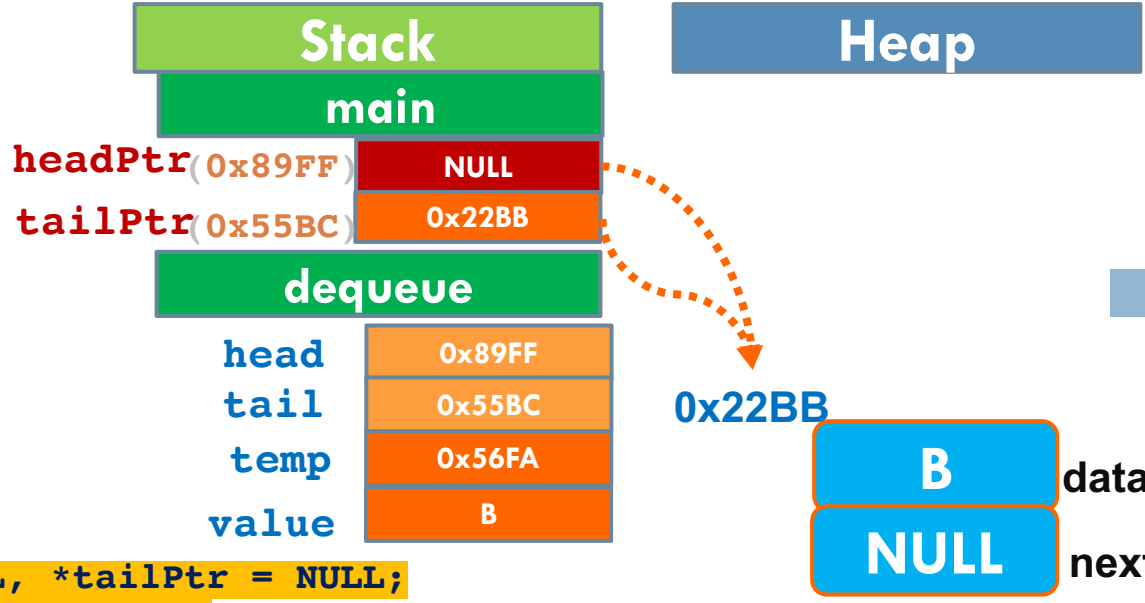

```c
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
      dequeue(&headPtr, &tailPtr);
    return 0;
}
char dequeue( QueueNode **head, QueueNode **tail ){
    char value;
    QueueNode *temp=*head;

    value = ( *head )->data;
    *head = ( *head )->next;

    if ( *head == NULL )
       *tail = NULL;

    free( temp );
    return value;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
    char data;
    struct node *next;
} QueueNode ;
```



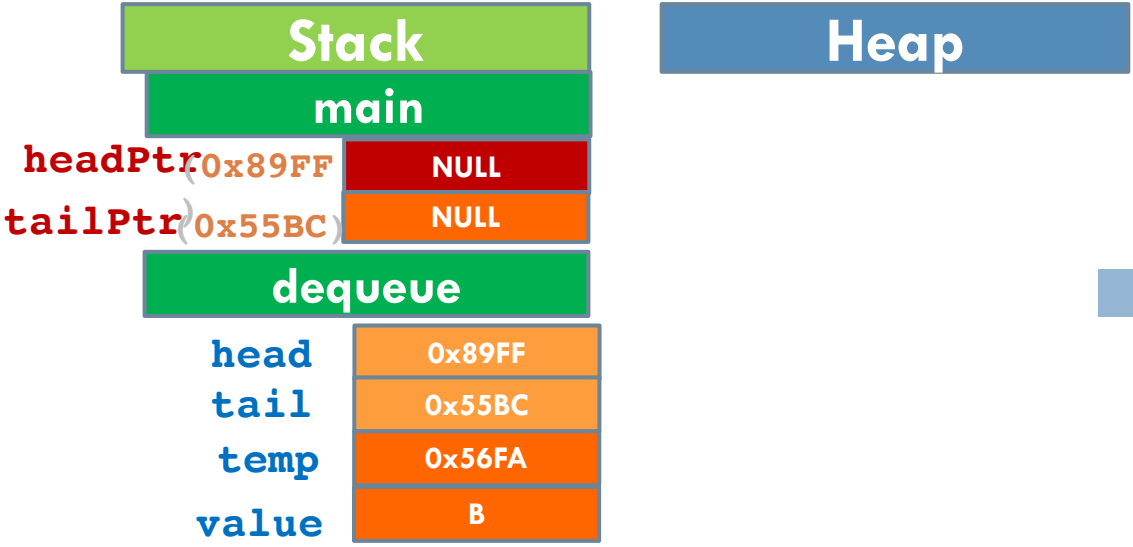

```c
int main()
{
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
      dequeue(&headPtr, &tailPtr);
    return 0;
}
char dequeue( QueueNode **head, QueueNode **tail ){
    char value;
    QueueNode *temp=*head;

    value = ( *head )->data;
    *head = ( *head )->next;

    if ( *head == NULL )
       *tail = NULL;

    free( temp );
    return value;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
   char data;
   struct node *next;
} QueueNode ;
```




```c
int main(){
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
      dequeue(&headPtr, &tailPtr);
      dequeue(&headPtr, &tailPtr);
    return 0;
}
char dequeue( QueueNode **head, QueueNode **tail ){
   char value;
   QueueNode *temp=*head;
   value = ( *head )->data;
   *head = ( *head )->next;

   if ( *head == NULL )
      *tail = NULL;

   free( temp );
   return value;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
   char data;
   struct node *next;
} QueueNode ;
```




```c
int main(){
      QueueNode *headPtr = NULL, *tailPtr = NULL;
      enqueue( &headPtr, &tailPtr, 'A');
      enqueue( &headPtr, &tailPtr, 'B');
      dequeue(&headPtr, &tailPtr);
      dequeue(&headPtr, &tailPtr);
    return 0;
}
char dequeue( QueueNode **head, QueueNode **tail ){
   char value;
   QueueNode *temp=*head;
   value = ( *head )->data;
   *head = ( *head )->next;

   if ( *head == NULL )
      *tail = NULL;

   free( temp );
   return value;
}
```

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

11η Εβδομάδα: Στοίβες, Ουρές και Αναδρομή

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.**

Αναδρομή

- Βασική έννοια στα Μαθηματικά και στην Πληροφορική.

- Στην πληροφορική η αναδρομή χρησιμοποιείται σαν *τεχνική προγραμματισμού* και σαν *μέθοδος σχεδιασμού αλγορίθμων*.

- Στον προγραμματισμό η αναδρομή εμφανίζεται με την **κλήση ενός υποπρογράμματος από τον εαυτό του**.

Αναδρομή

- Ένα αναδρομικό υποπρόγραμμα αποτελείται από:
  - ένα βήμα διακοπής:
    Για το οποίο γνωρίζουμε την απάντηση και μπορεί να λυθεί χωρίς να εφαρμόσουμε άλλες αναδρομικές κλήσεις της συνάρτησης.
    - ❖Κάθε αναδρομική συνάρτηση πρέπει να έχει τουλάχιστον ένα βήμα διακοπής!
  - ένα αναδρομικό βήμα:
    Το οποίο ορίζει το αποτέλεσμα λύσης ενός μεγαλύτερου προβλήματος σπάζοντας το σε μικρότερα κομμάτια. Για να πάρει τις λύσεις των «μικρότερων» υπο-προβλημάτων, καλεί ξανά την ίδια συνάρτηση.

Μη-αναδρομικές συναρτήσεις

Προτού δούμε τι είναι αναδρομή θεωρήστε το πρόβλημα εύρεσης του παραγοντικού κάποιου αριθμού (factorial).

0! *= 1 ,* ***1!*** *= 1* ***2!*** *= 1x2 = 2* ***3!*** *= 1x2x3 = 6*

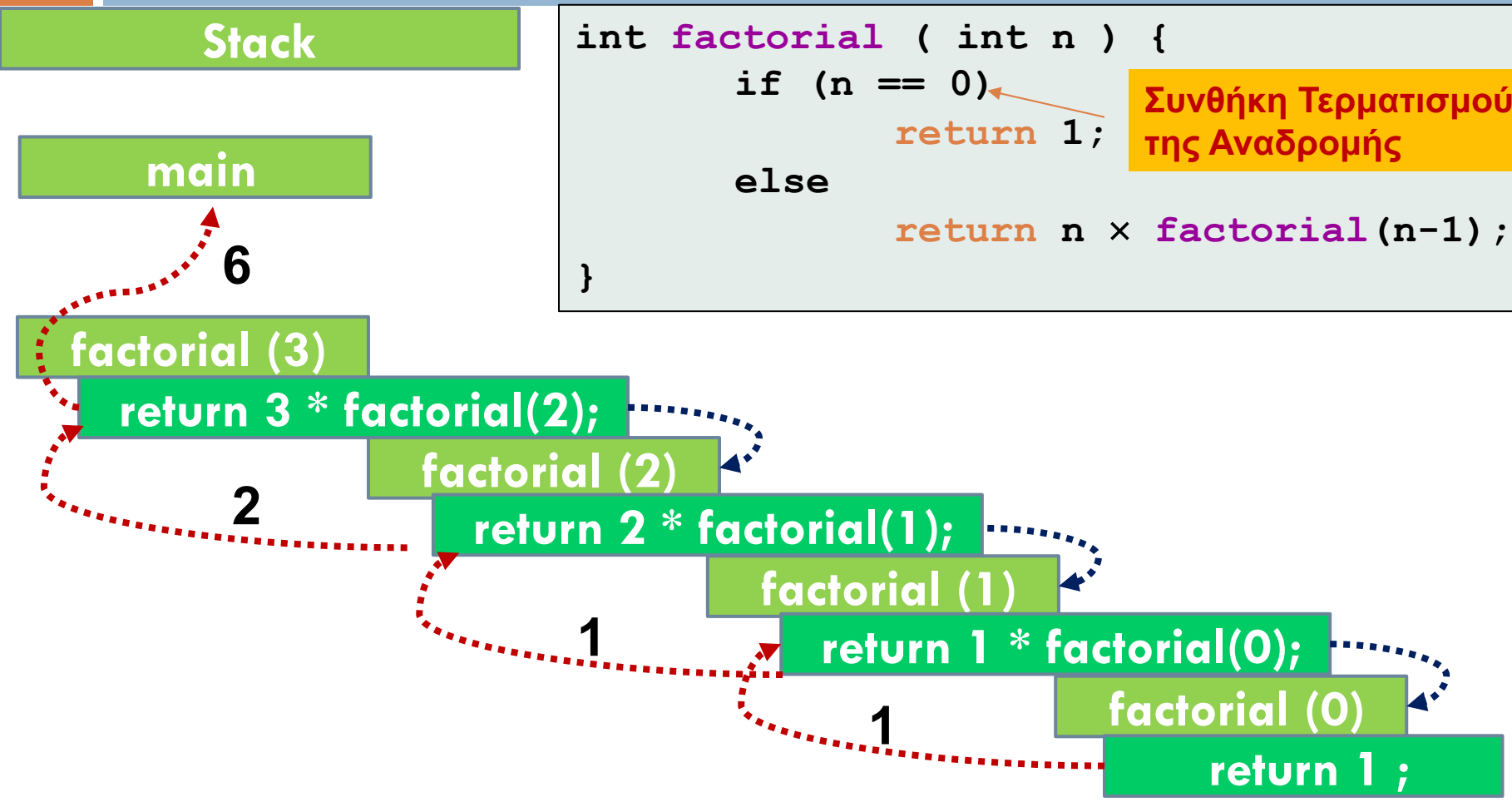
4! *= 1x2x3x4=24* ***5!*** *= 1x2x3x4x5 = 120*

Ζητούμενο: *Να υλοποιήσουμε την factorial(int n) η οποία μας επιστρέφει το παραγοντικό κάποιου θετικού ακεραίου n.*

Λύση

```
int factorial(int n) {
        int i, result=1;
        for (i=1; i<=n; i++) {
                result *= i;
        }
        return result;
}
```

Αναδρομή - Παραγοντικό




```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

Συνθήκη Τερματισμού της Αναδρομής

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Εκτέλεση factorial(3)

```
return 3 × factorial(2);
```

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Εκτέλεση factorial(3)

```
return 3 × factorial(2);
```

Εκτέλεση factorial(2)

```
return 2 × factorial(1);
```

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Εκτέλεση factorial(3)

```
return 3 × factorial(2);
```

Εκτέλεση factorial(2)

```
return 2 × factorial(1);
```

Εκτέλεση factorial(1)

```
return 1 × factorial(0);
```

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Εκτέλεση factorial(3)

```
return 3 × factorial(2);
```

Εκτέλεση factorial(2)

```
return 2 × factorial(1);
```

Εκτέλεση factorial(1)

```
return 1 × factorial(0);
```

Εκτέλεση factorial(0)

```
return 1;
```

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Εκτέλεση factorial(3)

```
return 3 × factorial(2);
```

Εκτέλεση factorial(2)

```
return 2 × factorial(1);
```

Εκτέλεση factorial(1)

```
return 1 × factorial(0);
```

1

Εκτέλεση factorial(0)

```
return 1;
```

Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

printf("%d", factorial(3))

Εκτέλεση factorial(3)

return 3 × factorial(2);

Εκτέλεση factorial(2)

return 2 × ~~factorial~~(1); 1

Εκτέλεση factorial(1)

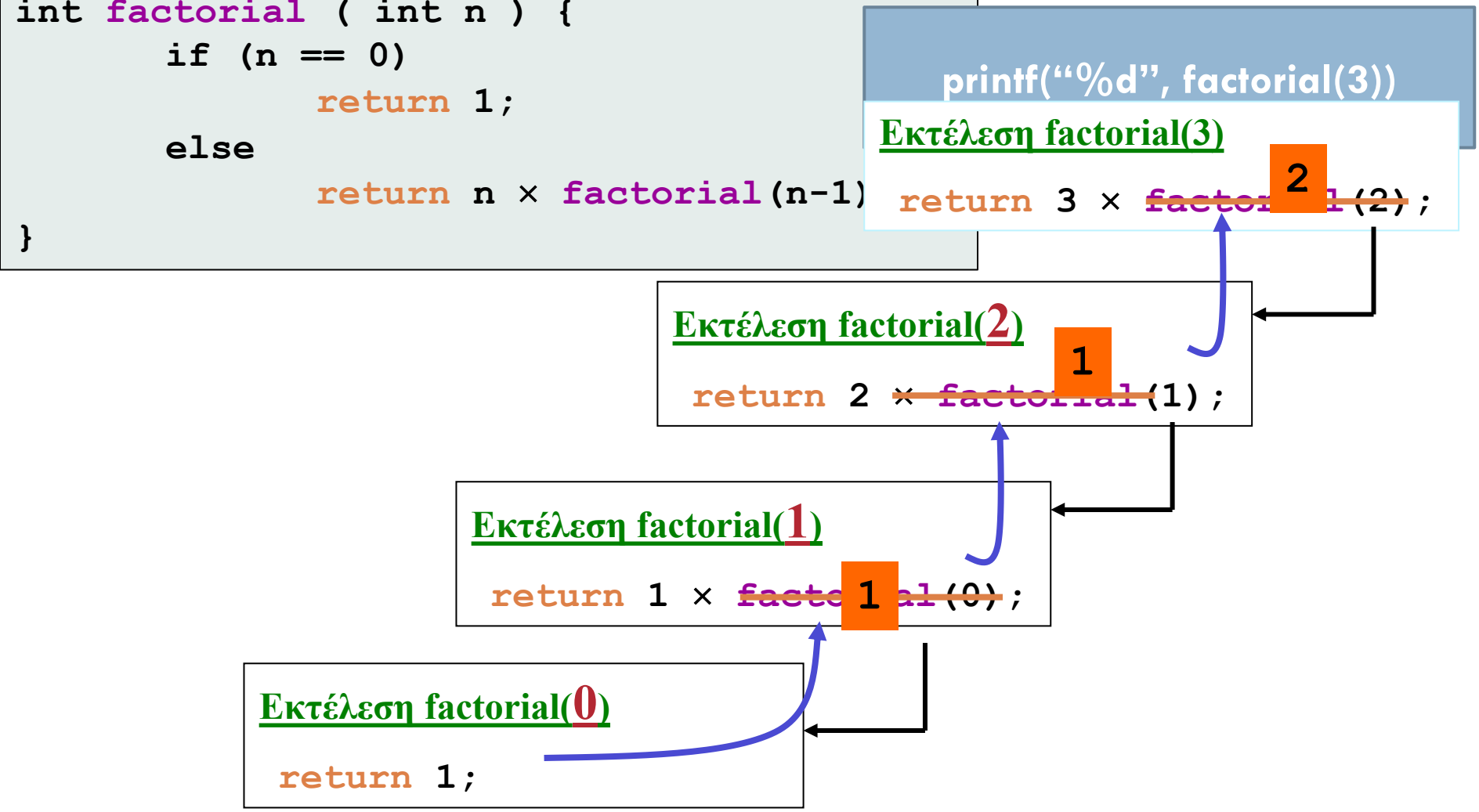
return 1 × ~~factorial~~(0); 1

Εκτέλεση factorial(0)

return 1;

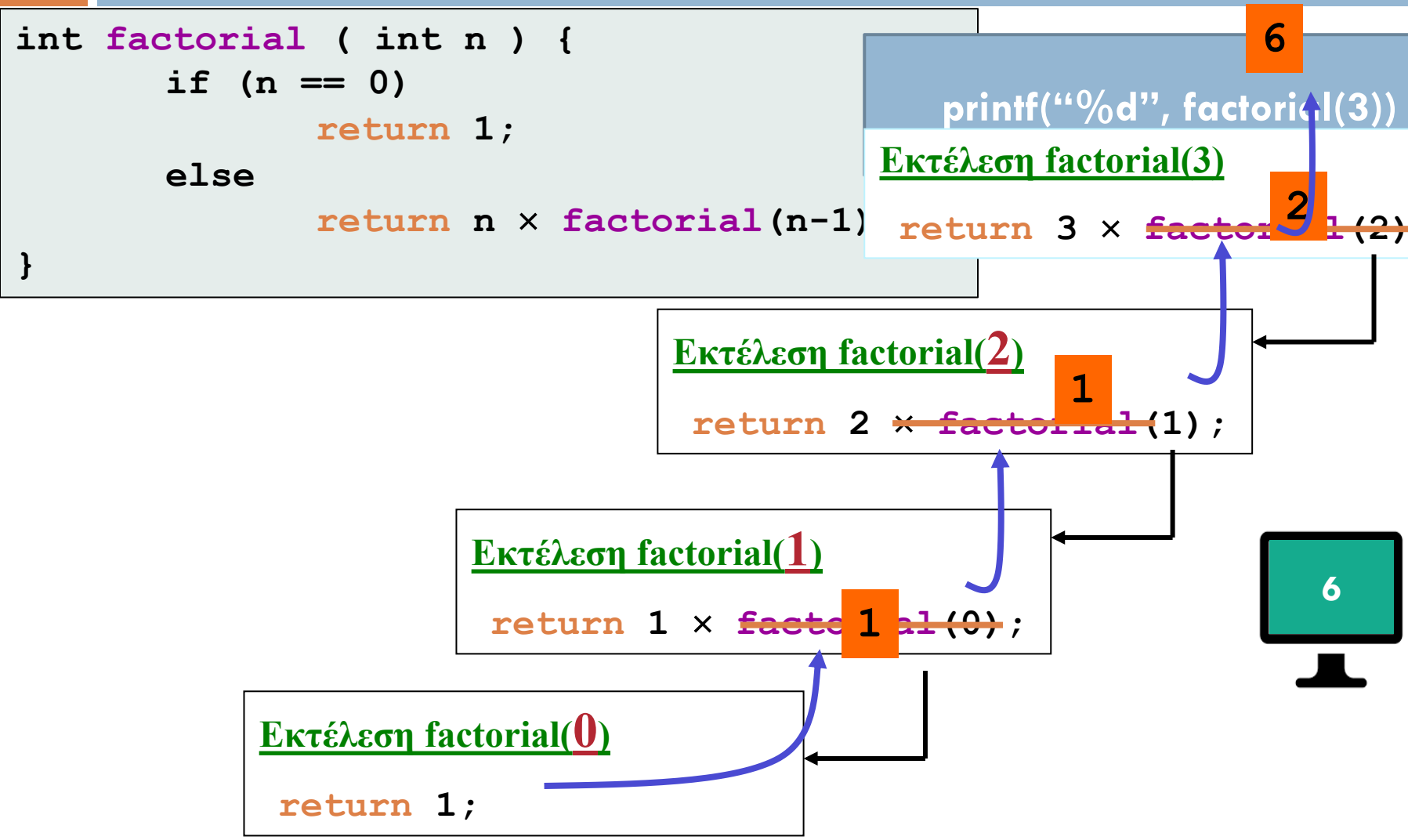
Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

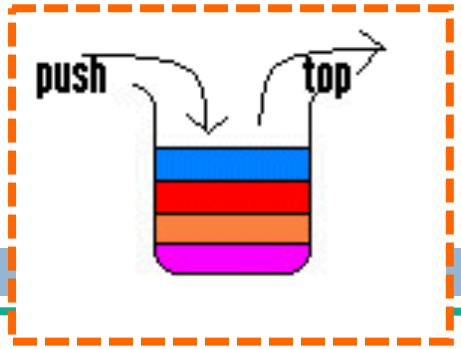


Αναδρομή - Παραγοντικό


```
int factorial ( int n ) {
        if (n == 0)
                return 1;
        else
                return n × factorial(n-1);
}
```

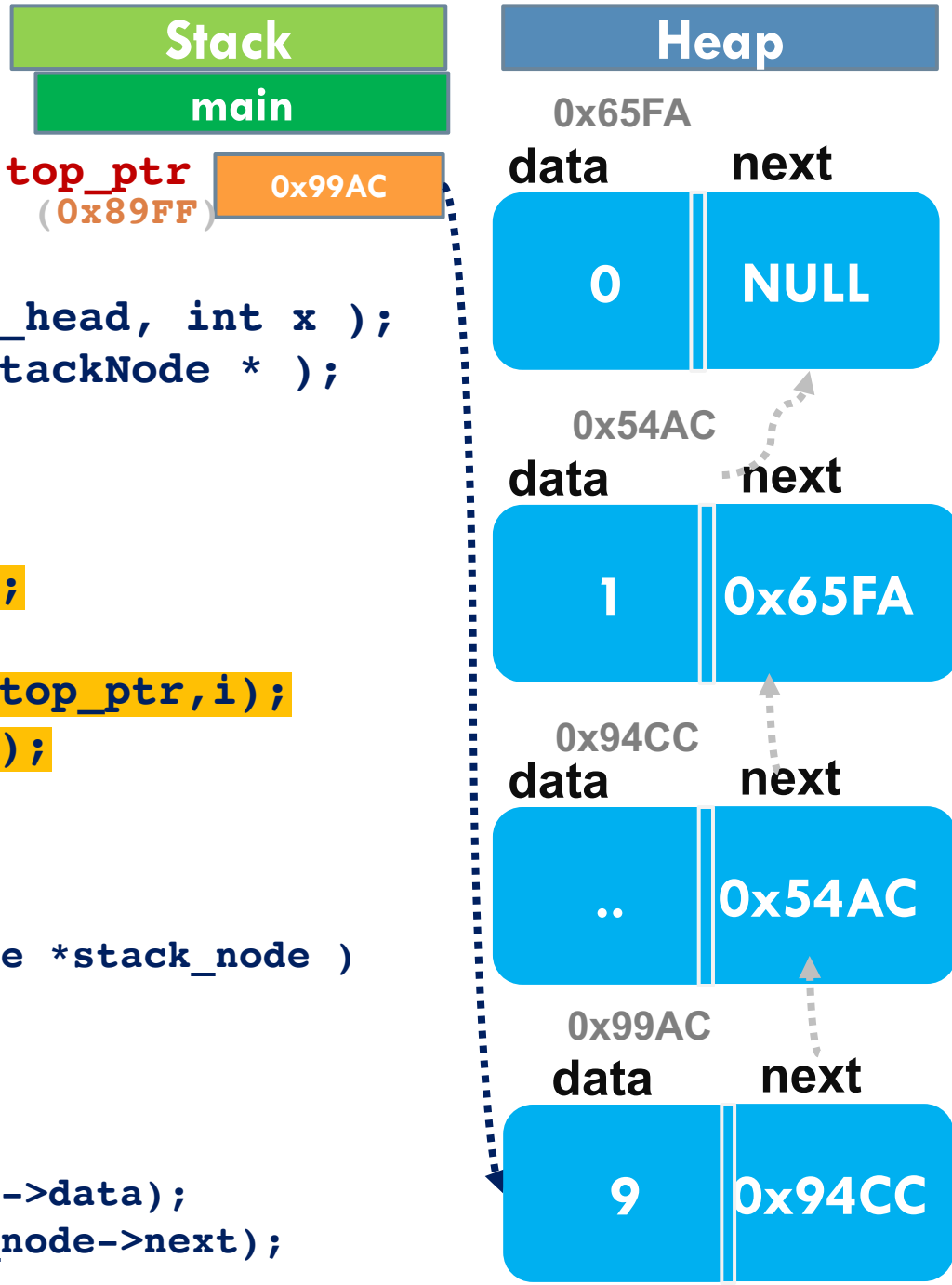


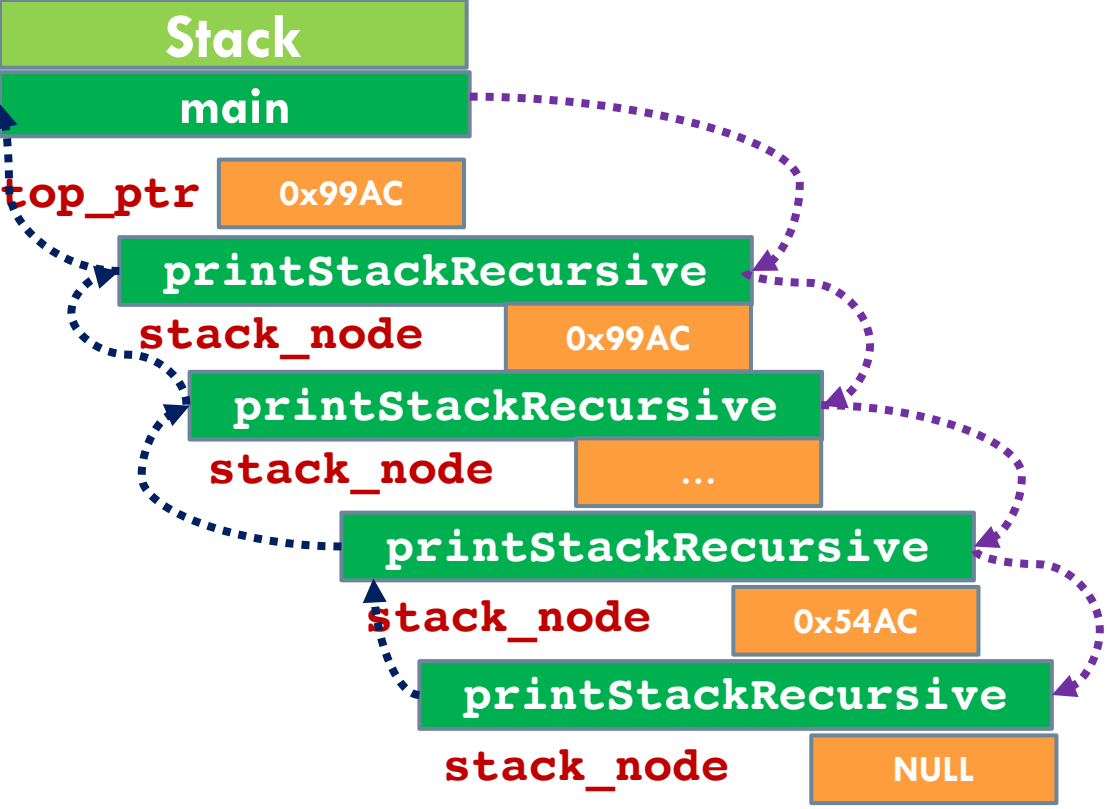
Εκτύπωση Στοίβας- Αναδρομή




```
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);
   }
```





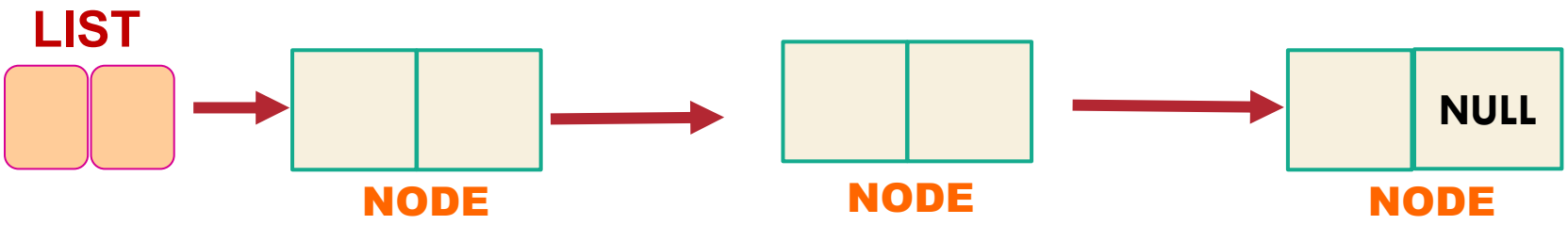

```
void printStackRecursive( StackNode *stack_node )
{
   if (stack_node == NULL )
      printf( "NULL\n\n" );
   else {
      printf("%d  -->", stack_node->data);
      printStackRecursive ( stack_node->nextPtr);
   }
   return;
}
```

Ευθύγραμμες Συνδεδεμένες Λίστες – Χρήση Αναδρομικών Συναρτήσεων

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

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

Μπορούμε μέσω της LIST να αναφερόμαστε στον πρώτο κόμβο της λίστας και να καταγράφουμε λοιπά χαρακτηριστικά της λίστας ανάλογα με το πρόβλημα (π.χ. size, numberof_odd_nodes, number_of_even_nodes, κτλ):



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

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

void createLinkedList(LIST *,int );
void printlist(LIST *);

int main()
{
    LIST mylist={NULL,0};
    createLinkedList(&mylist,10);
    printlist(&mylist);
    return 0;
}
```

```c
void createLinkedList(LIST * list,int x){
    NODE *tmp;
    int i=0;
    for(;i<x;i++){
          if(list->head==NULL){
             list->head=createNode(list, i);
             tmp=list->head;
          }
          else{
              tmp->next=createNode(list, i);
              tmp=tmp->next;
          }
    }
    return ;
}
```

```c
NODE* createNode (LIST * list, int value) {
        NODE *node;
        node = (NODE *)malloc(sizeof(NODE));
        if (node == NULL) {
            printf("Memory not allocated.\n");
            exit(0);
        }
        else{
          list->size=list->size+1;
          node->val = value;
          node->next = NULL;
          return node;
        }
}
```

Αναζήτηση Κόμβου σε Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

Με παρόμοιο τρόπο μπορούμε να ορίσουμε διαδικασία findpointer(LIST *L, key val) που επιστρέφει δείκτη προς κόμβο της λίστας που περιέχει το στοιχείο val, αν υπάρχει.

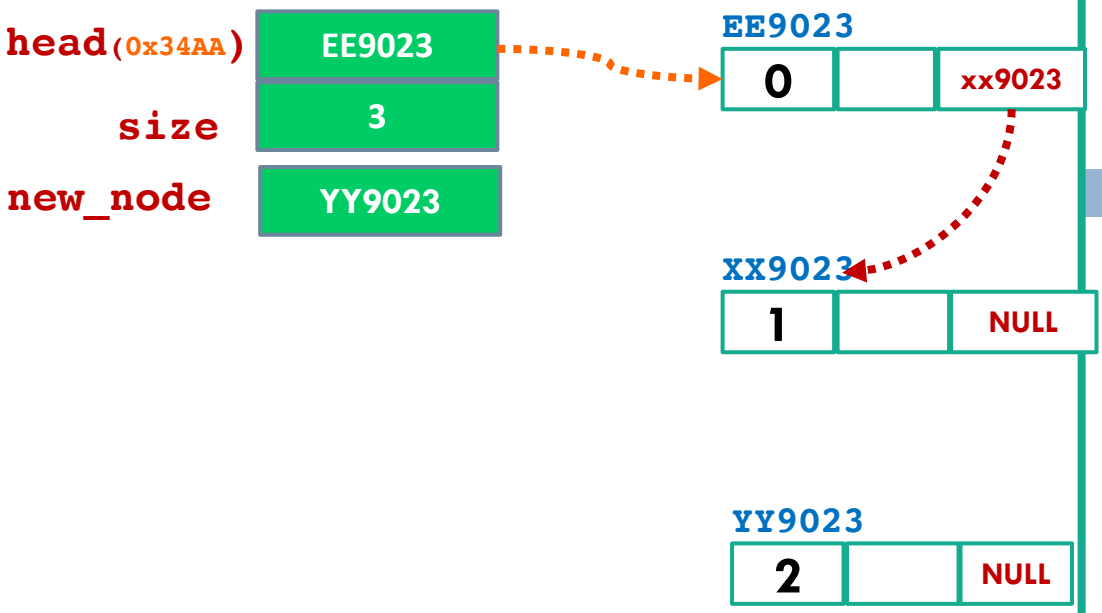
```
NODE *findpointer(LIST *list, int value) {
  NODE *p = list->head;
  while (p != NULL) {
    if (p->val == value) {
          return p;
    }
    p = p->next;
  }
  return NULL;
}
```

Αναζήτηση Κόμβου με Αναδρομή σε Ευθύγραμμες Απλά Συνδεδεμένες Λίστες

Με παρόμοιο τρόπο μπορούμε να ορίσουμε διαδικασία findpointer(LIST *L, key val) που επιστρέφει δείκτη προς κόμβο της λίστας που περιέχει το στοιχείο val, αν υπάρχει.

```
NODE *findpointer(LIST *list, int value) {

   NODE *p = list->head;

   if (p == NULL) return NULL;

   else if (p->val == value)  return p;

   return findpointer (p->next, value);
}
```

```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;
}
```

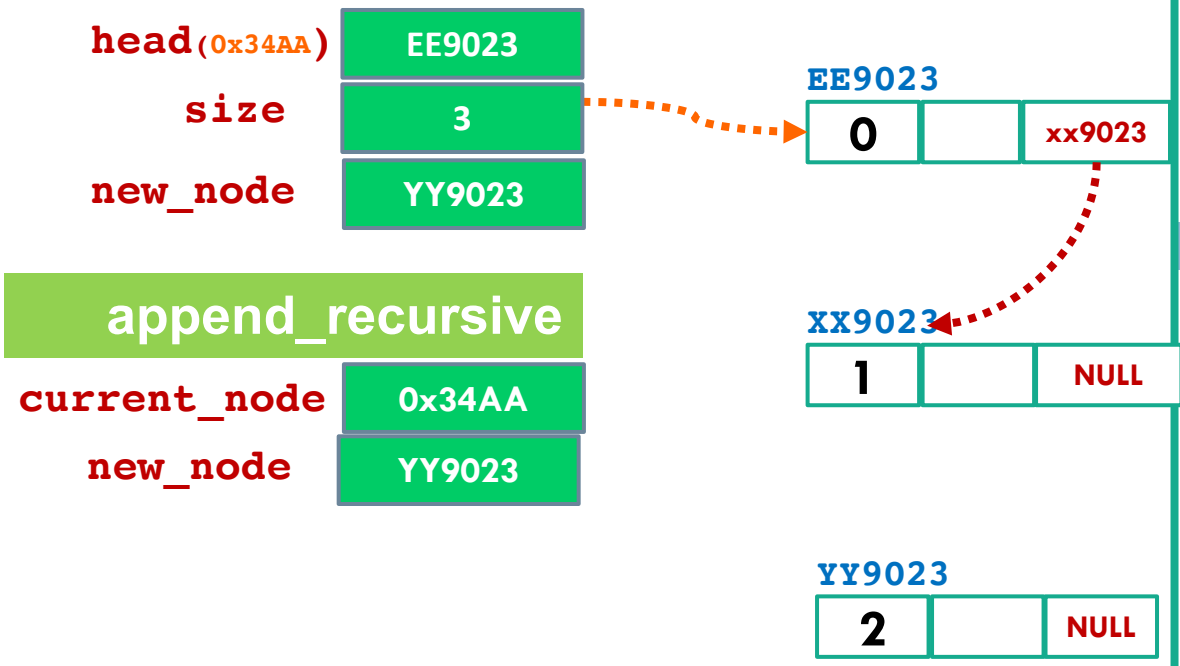


Προσθήκη Κόμβου με Αναδρομή στο τέλος Ευθύγραμμης Απλά Συνδεδεμένης Λίστας

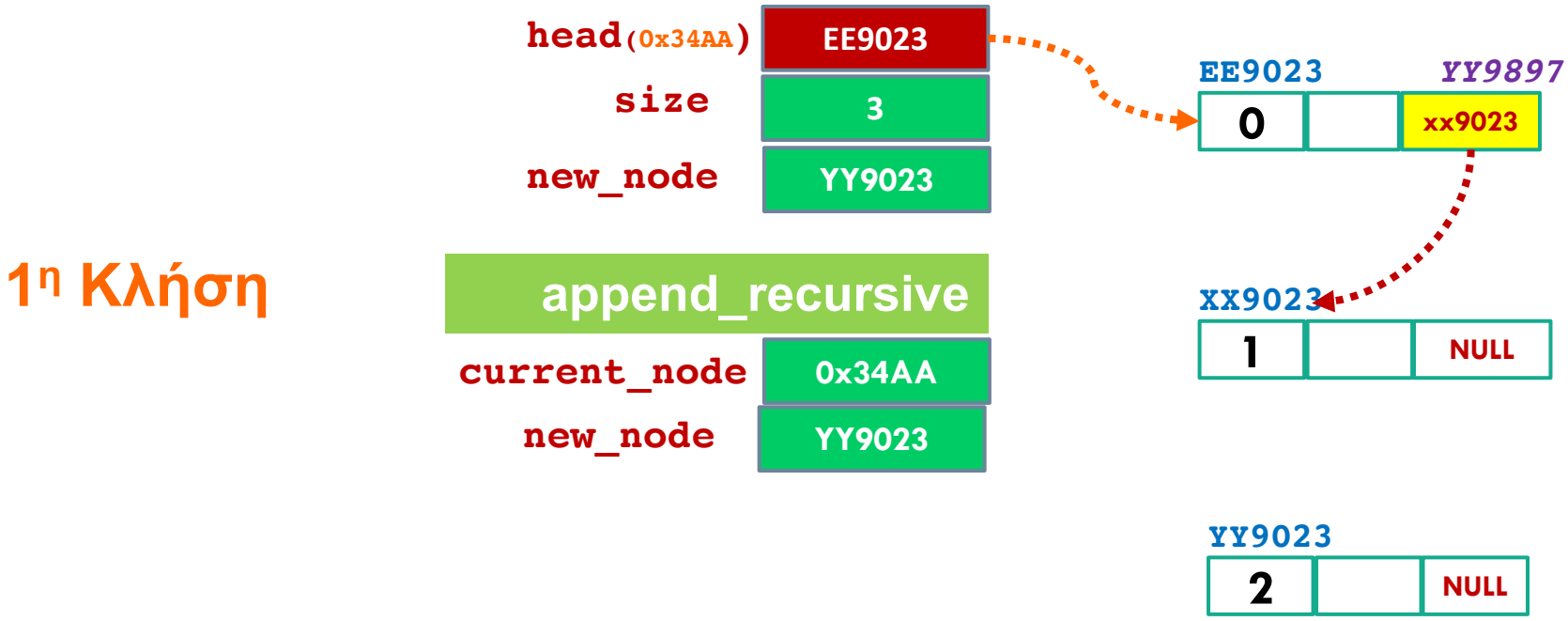
Προσθήκη Κόμβου με Αναδρομή στο τέλος Ευθύγραμμης Απλά Συνδεδεμένης Λίστας


```
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;

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;
}
```

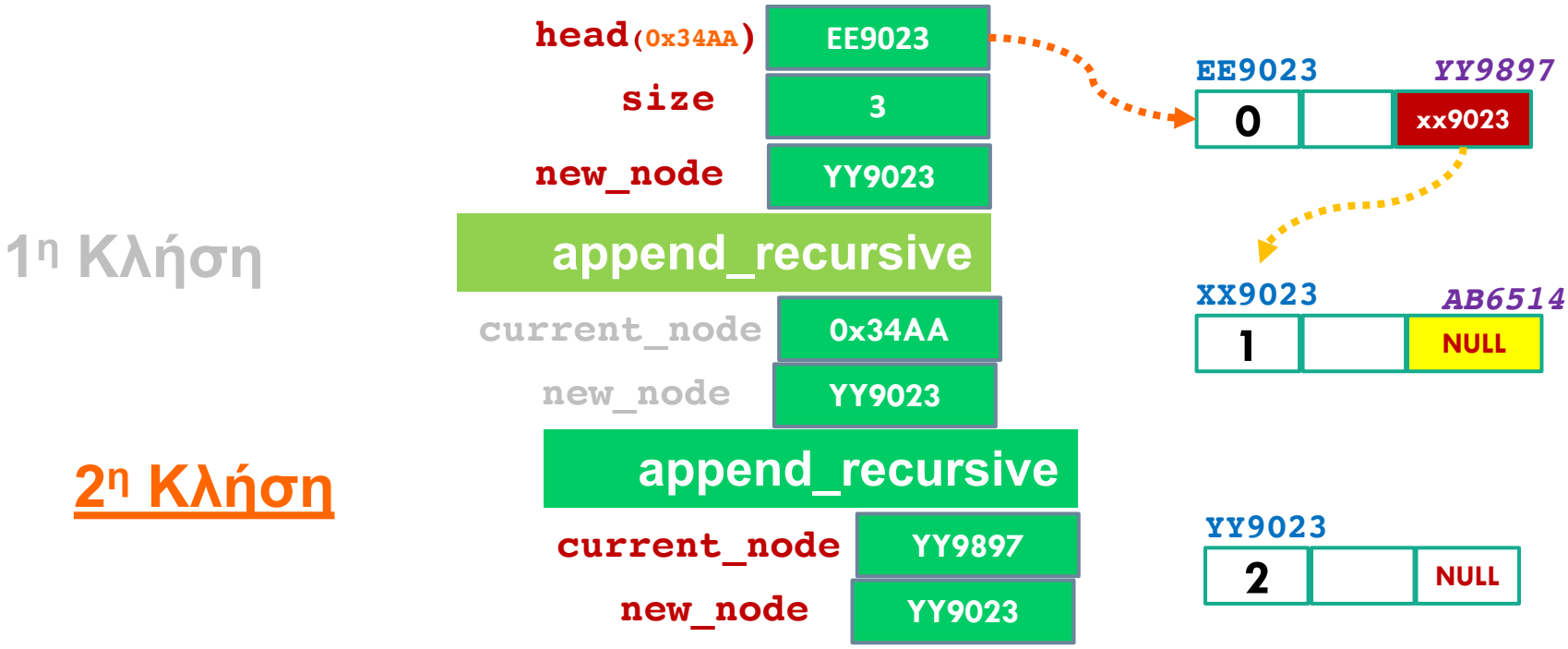



```c
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) {
            *current_node = new_node;
         return ;
        }
        append_recursive( & ( (*current_node)->next ), new_node );
      return;
}
```

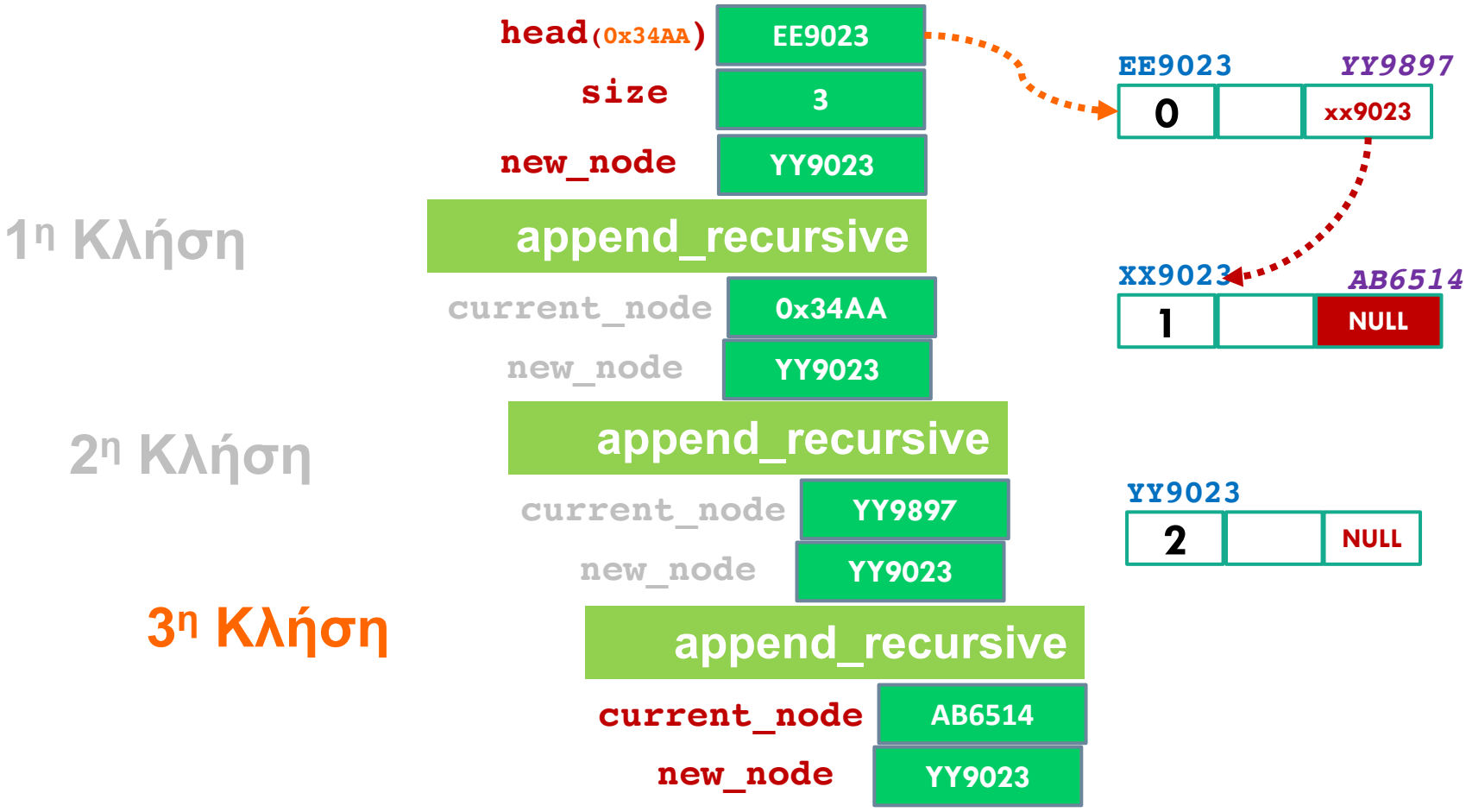


1η Κλήση

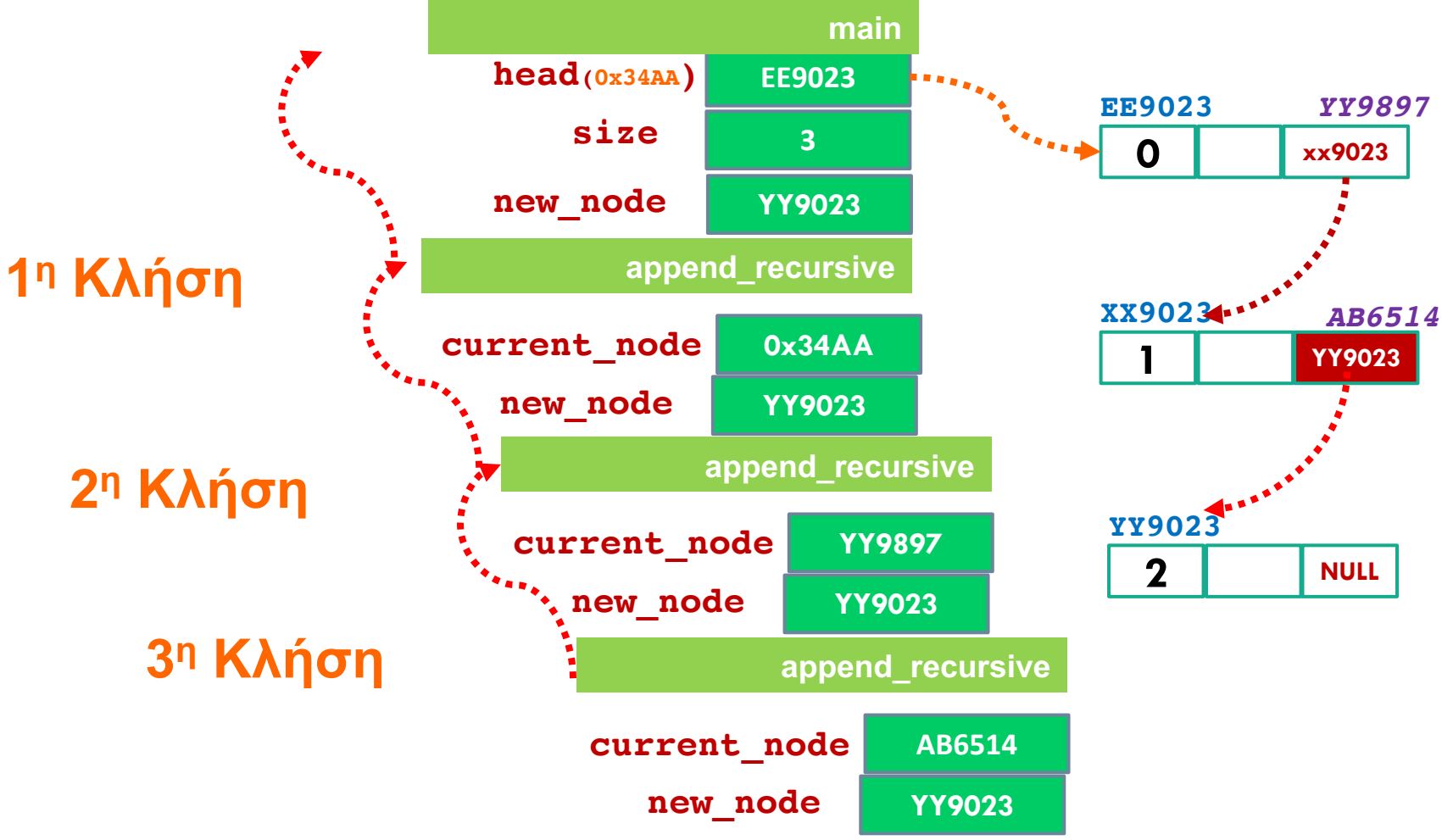
```
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) {
              *current_node = new_node;
         return ;
        }
        append_recursive( & ( (*current_node)->next ), new_node );
      return;
}
```




```
void append_recursive(NODE **current_node, NODE *new_node) {
        if (*current_node == NULL) { /* *curent_node is xx9023 */
              *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/