# Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

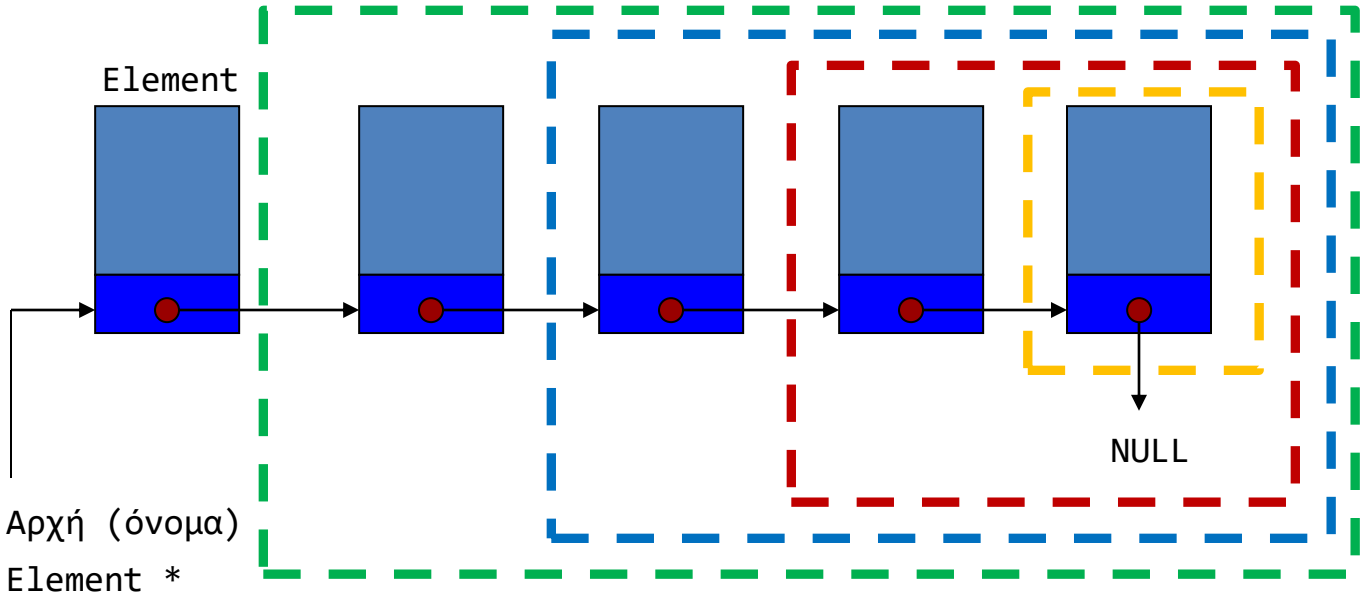
# Χρήσιμα σημεία σε συναρτήσεις επεξεργασίας λίστας

- Η λίστα ως παράμετρος
  - Αναφερόμαστε σε λίστα με τη διεύθυνση του πρώτου στοιχείου
  - Μερικές συναρτήσεις αλλάζουν τη διεύθυνση του πρώτου στοιχείου (προσθέτουν, διαγράφουν, …)
  - Άλλες όχι (εκτύπωση, καταμέτρηση, αναζήτηση…)
- Πρώτα χρησιμοποιώ, μετά διαγράφω
- Βρίσκω και καλύπτω ειδικές περιπτώσεις
  - Ενδεικτικά: άδεια λίστα, πρώτο στοιχείο, κενό στοιχείο ως είσοδος

# Αναδρομική αναζήτηση σε λίστα

```
#include <string.h>
#include "mytypes.h"

Listelement_ptr recfind(List alist, char name[]) {
if (alist == NULL)
   return NULL;
else
          if (!strcmp(alist->name, name))
             return alist;
          else
             return recfind(alist->next, name);
}
```

# Αυτοαναφορική (self-referential) δομή




Ένας κόμβος δείχνει σε κόμβο ίδιου τύπου (ή NULL)

Άλλη ανάγνωση: Κάθε κόμβος δείχνει σε μικρότερη λίστα (ή NULL)

# Απλοποίηση κώδικα

- Αξιοποίηση αυτοαναφορικής δομής
- Αναδρομική διαχείριση λίστας

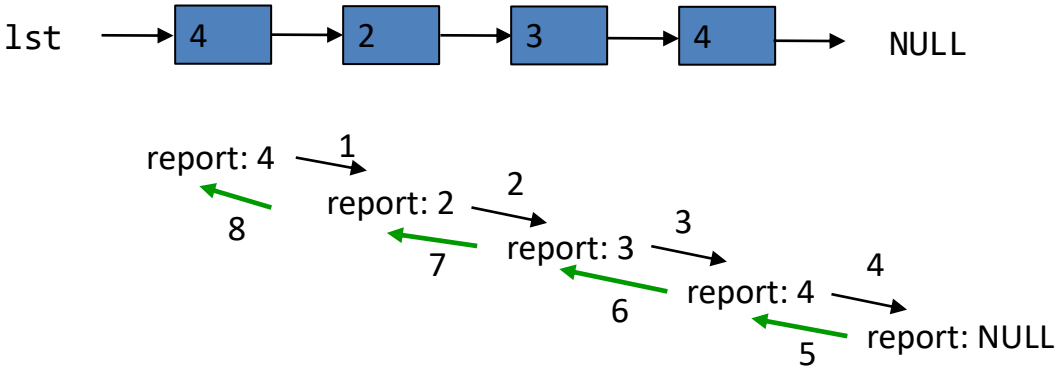
```
struct node {
          int value ;
          struct node * next;
};
typedef struct node Node;
typedef struct node * Node_ptr;
typedef struct node * List;
```

```
Node_ptr createnode(int a);
void report(List lst) ;
void append ( List * list_ptr, Node_ptr node_ref);
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "mylst.h"
#define N 10

int main(void ) {

        int i ;
        int temp ;
        List mylst = NULL;
        Node_ptr node_ref;

        srand(0);

        for (i=0; i<N; i++) {
                temp = rand() % 10 ;
                node_ref = createnode(temp);
                append( &mylst, node_ref) ;
        }

        report(mylst);

        return EXIT_SUCCESS;
}
```

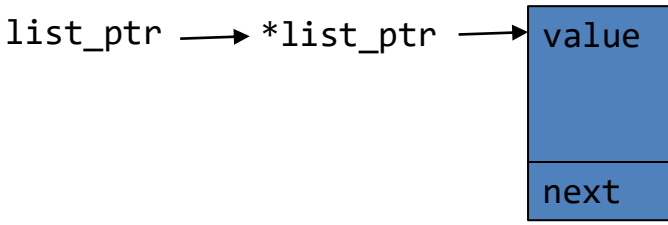
```c
#include <stdio.h>
#include <stdlib.h>
#include "mylst.h"

Node_ptr createnode(int a) {
          Node_ptr new_node_ptr;

          new_node_ptr = malloc( sizeof (Node) ) ;
          new_node_ptr -> value = a;
          new_node_ptr -> next = NULL;

          return new_node_ptr;

}
```

```
void report(List lst) {

        if (lst == NULL) return;

        printf("%d ", lst -> value);
        report ( lst->next);

        return ;
}
```




Να θυμηθούμε το stack, recursion depth limits

```
void append ( List * list_ptr, Node_ptr node_ref) {

        if ( *list_ptr == NULL) {
                *list_ptr = node_ref;
                return ;
        };

        append( &( (*list_ptr)->next ), node_ref );

        return ;
}
```




(*list_ptr)->next είναι lvalue

Βρίσκεται στη διεύθυνση &((*list_ptr)->next)

# Αναδρομή και pass by value

```
List append ( List list, Node_ptr node_ref) {

          if ( list == NULL) {
                    return node_ref;
          };

          list->next = append(list->next , node_ref );

          return list;
}
```

- Αφαιρετικότητα στα δεδομένα
  - Έλεγχος τύπων
  - Δομές
  - Κατασκευή τύπων
- Δυναμική διαχείριση μνήμης
  - free(), malloc(), realloc(),…
- Αναδρομικός ορισμός συναρτήσεων

# Αναδρομική αναζήτηση σε λίστα

```
#include <string.h>
#include "mytypes.h"

Listelement_ptr recfind(List alist, char name[]) {
if (alist == NULL)
   return NULL;
else
       if (!strcmp(alist->name, name))
          return alist;
       else
          return recfind(alist->next, name);
}
```

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

typedef struct x {
           int a;
           struct x * next; } X;
typedef X * X_ptr;
typedef X * List;

X_ptr createnode (int a) {
    X_ptr temp ;
    temp = malloc(sizeof (X_ptr));
    temp ->   a = a;
    temp ->next = NULL;
    return temp;
}


void append( List * x, X_ptr a) {
    if (*x == NULL) {
        *x = a;
        return ;
        }
    append (& ( ((*x)->next)), a);
    return ;
}

List mymain () {
    static List y  = NULL;
    X_ptr a ;
    int x;
    scanf("%d", x);
    a = createnode(1);

    append(&y, a);
    return y;
}
```

Μερικές αναδρομικές συναρτήσεις είναι tail recursive:

Δεν συνδράμει στη λύση το trace back

```
void append( List * x, X_ptr a) {
    if (*x == NULL) {
        *x = a;
        return ;
        }
    append (& ( ((*x)->next)), a);
    return ;
}
```

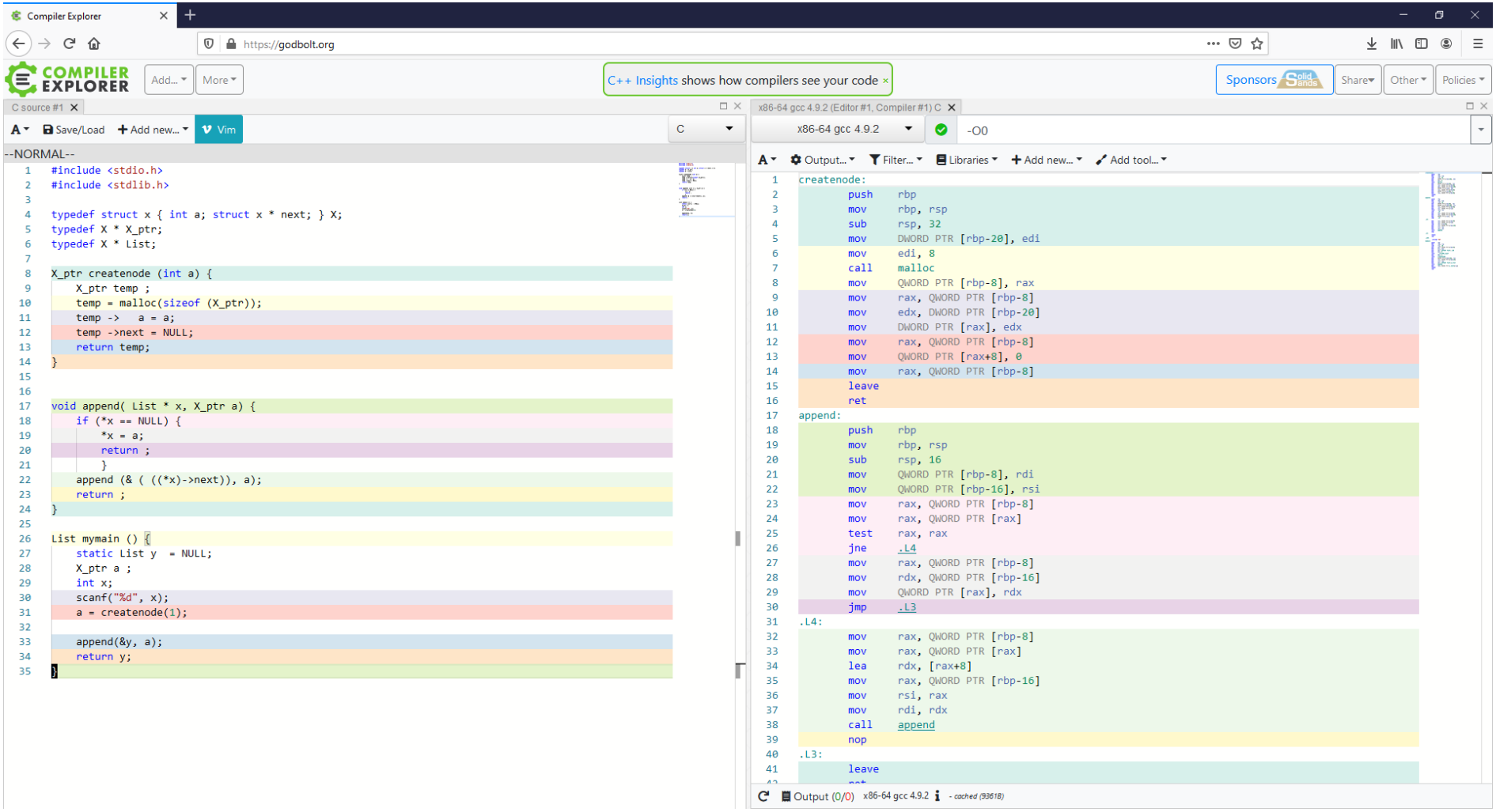
```
void append( List * x, X_ptr a) {
    while (*x != NULL) {
        *x = (*x) -> next;
        }
    *x = a;
    return ;
}
```

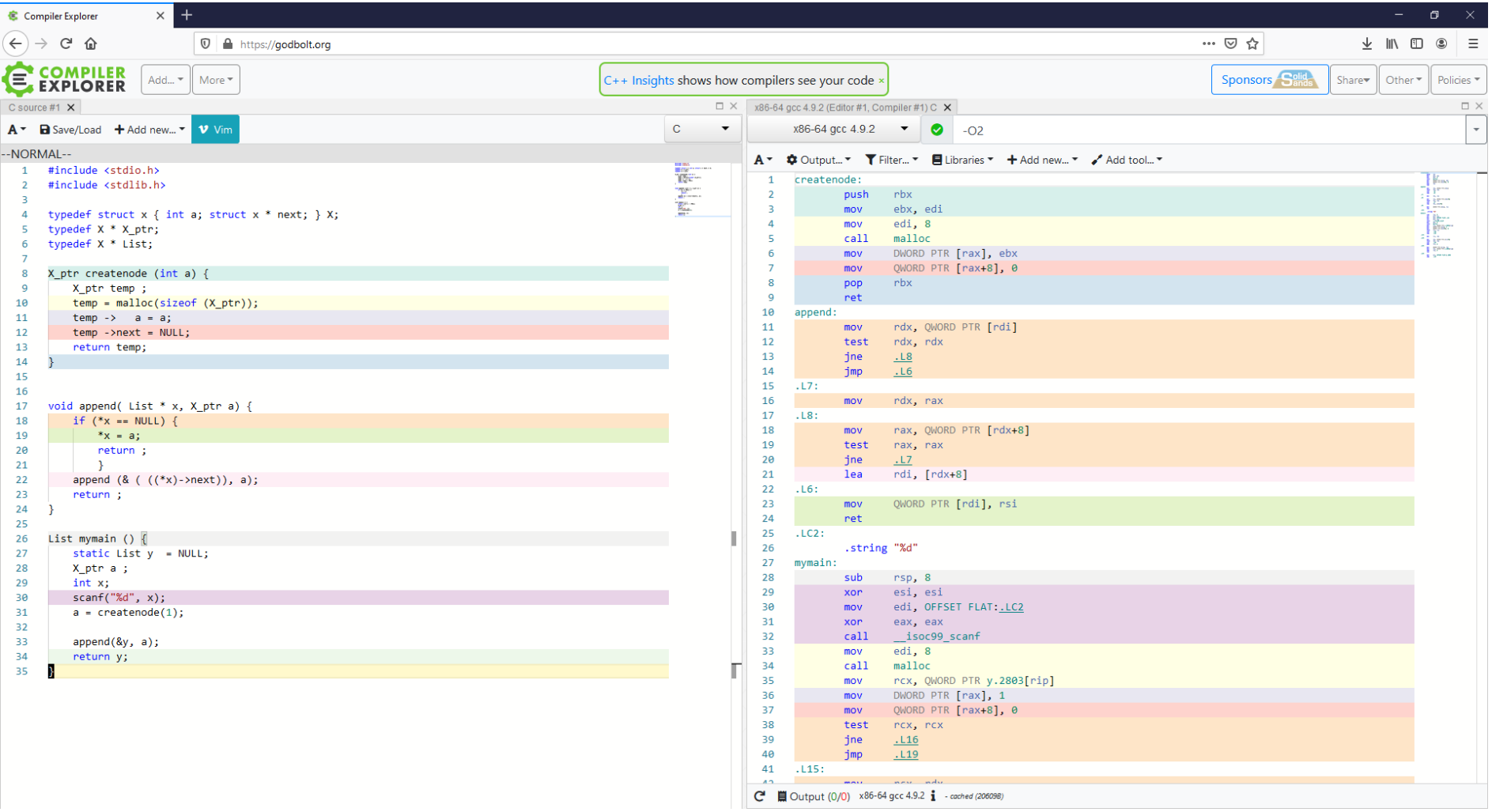
Μερικές αναδρομικές συναρτήσεις είναι tail recursive:
Δεν συνδράμει στη λύση το trace back

Η αναδρομική κλήση είναι το τελευταίο πράγμα πριν το return ή είναι η μόνη έκφραση του return (πχ return f(x-1);)

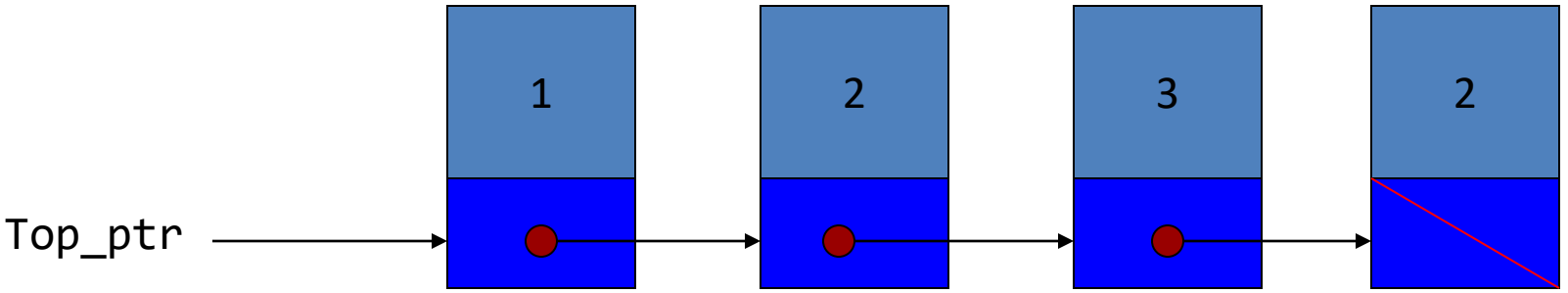
Μετασχηματίζονται σε επαναληπτικές λύσεις

Οι compilers εφαρμόζουν τέτοιες μετατροπές.


Compiler Explorer
https://godbolt.org
COMPILER EXPLORER
Add...
More
C++ Insights shows how compilers see your code
Sponsors
Share
Other
Policies
C source #1
Save/Load
Add new...
Vim
C
--NORMAL--
#include <stdio.h>
#include <stdlib.h>
typedef struct x { int a; struct x * next; } X;
typedef X * X_ptr;
typedef X * List;
X_ptr createnode (int a) {
X_ptr temp ;
temp = malloc(sizeof (X_ptr));
temp -> a = a;
temp ->next = NULL;
return temp;
}
void append( List * x, X_ptr a) {
if (*x == NULL) {
*x = a;
return ;
}
append (& ( ((*x)->next)), a);
return ;
}
List mymain () {
static List y = NULL;
X_ptr a ;
int x;
scanf("%d", x);
a = createnode(1);
append(&y, a);
return y;
}
x86-64 gcc 4.9.2 (Editor #1, Compiler #1) C
x86-64 gcc 4.9.2
-O0
Output...
Filter...
Libraries
Add new...
Add tool...
createnode:
push rbp
mov rbp, rsp
sub rsp, 32
mov DWORD PTR [rbp-20], edi
mov edi, 8
call malloc
mov QWORD PTR [rbp-8], rax
mov rax, QWORD PTR [rbp-8]
mov edx, DWORD PTR [rbp-20]
mov DWORD PTR [rax], edx
mov rax, QWORD PTR [rbp-8]
mov QWORD PTR [rax+8], 0
mov rax, QWORD PTR [rbp-8]
leave
ret
append:
push rbp
mov rbp, rsp
sub rsp, 16
mov QWORD PTR [rbp-8], rdi
mov QWORD PTR [rbp-16], rsi
mov rax, QWORD PTR [rbp-8]
mov rax, QWORD PTR [rax]
test rax, rax
jne .L4
mov rax, QWORD PTR [rbp-8]
mov rdx, QWORD PTR [rbp-16]
mov QWORD PTR [rax], rdx
jmp .L3
.L4:
mov rax, QWORD PTR [rbp-8]
mov rax, QWORD PTR [rax]
lea rdx, [rax+8]
mov rax, QWORD PTR [rbp-16]
mov rsi, rax
mov rdi, rdx
call append
nop
.L3:
leave
Output (0/0) x86-64 gcc 4.9.2 - cached (93618)

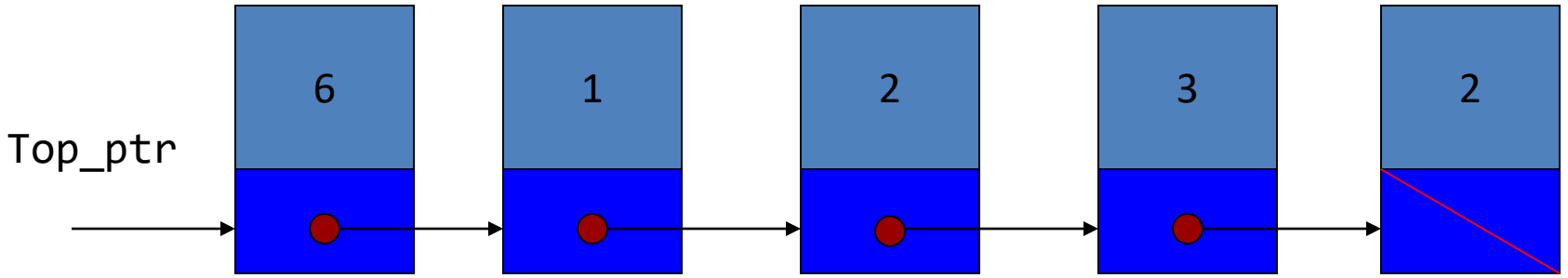

Compiler Explorer
https://godbolt.org
COMPILER EXPLORER
Add...
More
C++ Insights shows how compilers see your code
Sponsors
Share
Other
Policies
C source #1
Save/Load
Add new...
Vim
C
--NORMAL--
#include <stdio.h>
#include <stdlib.h>
typedef struct x { int a; struct x * next; } X;
typedef X * X_ptr;
typedef X * List;
X_ptr createnode (int a) {
X_ptr temp ;
temp = malloc(sizeof (X_ptr));
temp -> a = a;
temp ->next = NULL;
return temp;
}
void append( List * x, X_ptr a) {
if (*x == NULL) {
*x = a;
return ;
}
append (& ( ((*x)->next)), a);
return ;
}
List mymain () {
static List y = NULL;
X_ptr a ;
int x;
scanf("%d", x);
a = createnode(1);
append(&y, a);
return y;
}
x86-64 gcc 4.9.2 (Editor #1, Compiler #1) C
x86-64 gcc 4.9.2
-O2
Output...
Filter...
Libraries
Add new...
Add tool...
createnode:
push rbx
mov ebx, edi
mov edi, 8
call malloc
mov DWORD PTR [rax], ebx
mov QWORD PTR [rax+8], 0
pop rbx
ret
append:
mov rdx, QWORD PTR [rdi]
test rdx, rdx
jne .L8
jmp .L6
.L7:
mov rdx, rax
.L8:
mov rax, QWORD PTR [rdx+8]
test rax, rax
jne .L7
lea rdi, [rdx+8]
.L6:
mov QWORD PTR [rdi], rsi
ret
.LC2:
.string "%d"
mymain:
sub rsp, 8
xor esi, esi
mov edi, OFFSET FLAT:.LC2
xor eax, eax
call __isoc99_scanf
mov edi, 8
call malloc
mov rcx, QWORD PTR y.2803[rip]
mov DWORD PTR [rax], 1
mov QWORD PTR [rax+8], 0
test rcx, rcx
jne .L16
jmp .L19
.L15:
Output (0/0) x86-64 gcc 4.9.2
- cached (206098)

# Στοίβες - stacks

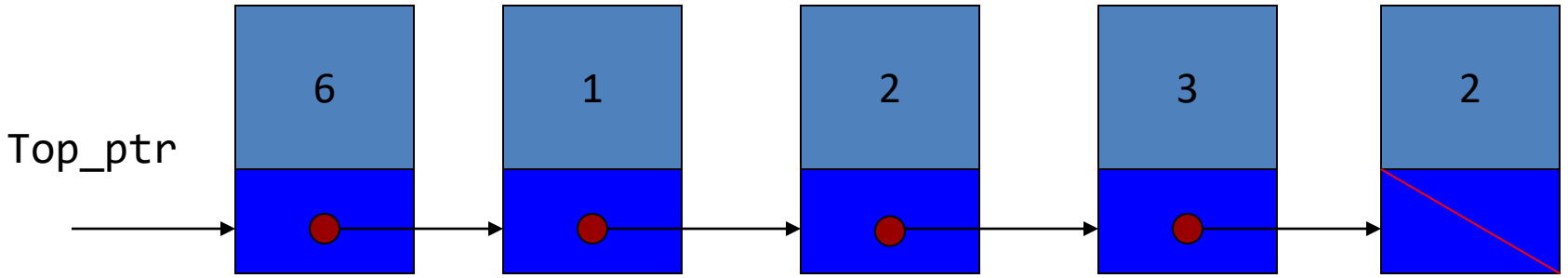



- Λειτουργίες push/pop
- Last-In First-Out (LIFO)

# Στοίβες – stacks: push



# Στοίβες – stacks: pop



# Συμβολισμός προθέματος
# Prefix notation

- `2*(5*(5 + 2) + 3*5)` `[= 100]`
- `2*(5*(+ 5 2) + *3 5)`
- `2*((* 5 + 5 2) + * 3 5)`
- `2*(+ * 5 + 5 2  * 3 5)`
- `  *2 +* 5 + 5 2 * 3 5`

- Διατηρώντας την προτεραιότητα, αντικαθιστώ κάθε έκφραση α ◊ β με ◊ α β
- Ο τελεστής προηγείται των τελεστέων
- Η τελική έκφραση *δε χρειάζεται παρενθέσεις!*

# Περιεχόμενα stack σε κάθε βήμα:

* 2 + * 5 + 5 2 * 3 5

Διατρέχω την έκφραση από
***δεξιά*** προς ***αριστερά***.

όταν βρίσκω αριθμό
    τον τοποθετώ στο σωρό
όταν βρίσκω τελεστή, τον εφαρμόζω
    στην κορυφή του σωρού
    και τοποθετώ το αποτέλεσμα
    στο σωρό

5

3 5

15

2 15

5 2 15

7 15

5 7 15

35 15

50

2 50

100

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

struct stackNode {/* self-referential
                     structure */
   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 );
```

```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;
}
```

```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 );
}
```

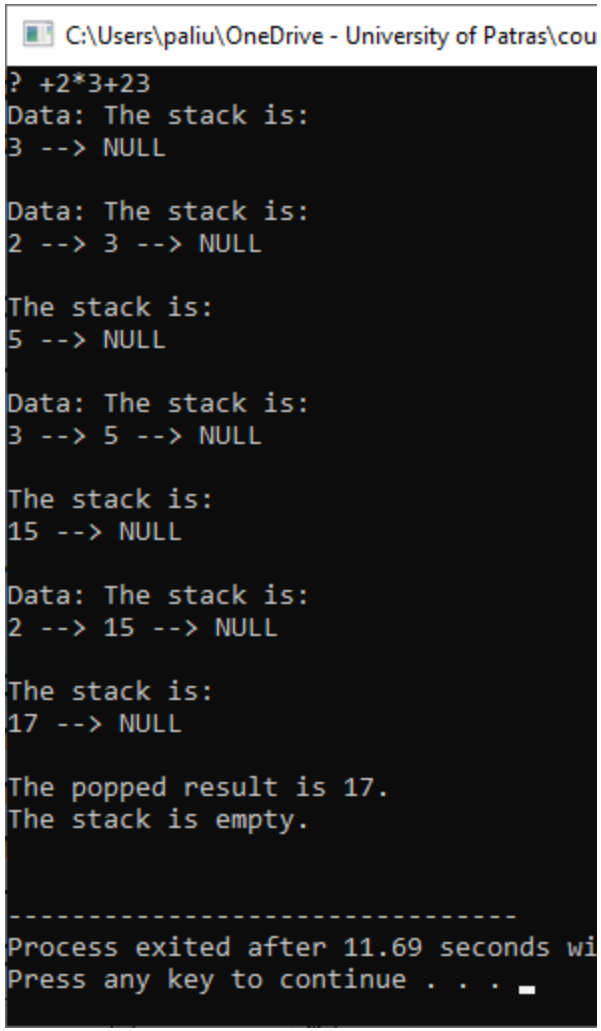
```c
/* Print the stack */
void printStack( StackNodePtr currentPtr )
{
   if ( currentPtr == NULL )
      printf( "The stack is empty.\n\n" );
   else {
      printf( "The stack is:\n" );

      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;
}
```

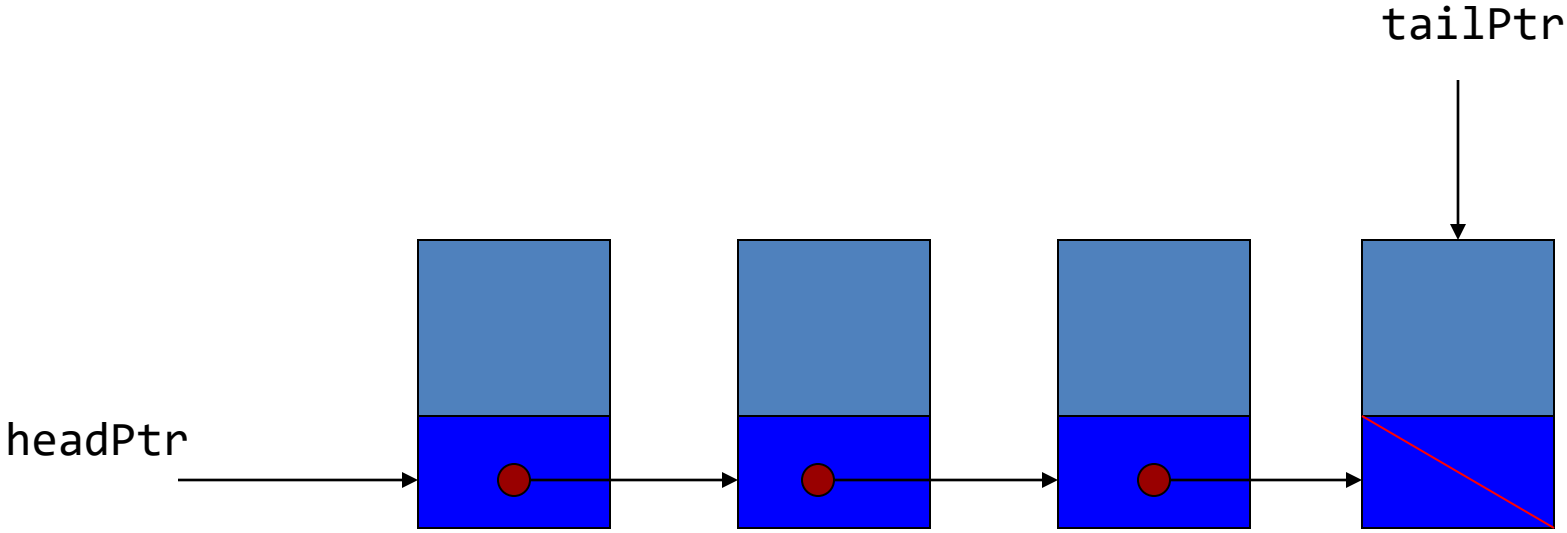
# Παράδειγμα: Υπολογισμός έκφρασης προθέματος

- Χρησιμοποιώντας ένα stack, να γράψετε ένα πρόγραμμα που να υπολογίζει μια έκφραση προθέματος.
- *Προσχέδιο* λύσης
  - Υλοποιεί τον αλγόριθμο της διαφάνειας 6
  - Η είσοδος είναι ένα αλφαριθμητικό χωρίς κενά, με μονοψήφια όχι προσημασμένα δεδομένα, χωρίς ελέγχους,…

```c
int main(void)
{
   StackNodePtr stackPtr = NULL;  /* points to stack top */
   int choice, value, i;
   int a, b;
   char expression[100];
   printf( "? " );
   scanf( "%99s", expression ); /* Can you explain %99s ? */

   for(i=strlen(expression)-1; i>=0; i--) {    /* read from right to left */
               choice = expression[i];
               if (isdigit(choice)) {          /* a single char only!!! */
                         value = choice - 48; /* Can you explain why? */
                         /* push data value onto stack */
                         printf("Data: ");
                         push( &stackPtr, value );
                         printStack( stackPtr );
                         }
                else {
                         a = pop(&stackPtr);
                         b = pop(&stackPtr);
                         switch(choice) {
                                   case '+': push(&stackPtr, a + b);
                                             printStack( stackPtr );
                                             break;
                                   case '*': push(&stackPtr, a * b);
                                             printStack( stackPtr );
                                             break;
                         }
                 }
   }
   /* pop result value off stack */
   if ( !isEmpty( stackPtr ) )
        printf( "The popped result is %d.\n", pop( &stackPtr ) );

   printStack( stackPtr );

   return EXIT_SUCCESS;
}
```



# Ουρές - queues




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

# Enqueue/Dequeue

- Enqueue: Στοιχεία τοποθετούνται στο tail

```
(head) A → B → C → NULL          (tail) ήρθε C
(head) A → B → C → D → NULL      (tail) ήρθε D
(head) A → B → C → D → E → NULL (tail) ήρθε E
```

- Dequeue: Στοιχεία αποχωρούν από head

```
εξυπηρετείται το A   (head)     B → C → D → E → NULL   (tail)
εξυπηρετείται το B   (head)         C → D → E → NULL   (tail)
```

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

struct queueNode {   /* self-referential structure */
   char data;
   struct queueNode *nextPtr;
};

typedef struct queueNode QueueNode;
typedef QueueNode *QueueNodePtr;

/* function prototypes */
void printQueue( QueueNodePtr );
int isEmpty( QueueNodePtr );
char dequeue( QueueNodePtr *, QueueNodePtr * );
void enqueue( QueueNodePtr *, QueueNodePtr *, char );
void instructions( void );
```

```c
void instructions( void )
{
   printf ( "Enter your choice:\n"
            "   1 to add an item to the queue\n"
            "   2 to remove an item from the queue\n"
            "   3 to end\n" );
}
```

```c
int main(void) {
   QueueNodePtr headPtr = NULL,
                tailPtr = NULL;
   int choice;
   char item;

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

   while ( choice != 3 ) {
      switch( choice ) {
       case 1:
         printf( "Enter a character: " );
         scanf( "\n%c", &item );
            enqueue( &headPtr, &tailPtr, item);
            printQueue( headPtr );
            break;

case 2:
        if ( !isEmpty( headPtr ) ) {
          item = dequeue( &headPtr, &tailPtr );
          printf( "%c has been dequeued.\n", item );
          }
         printQueue( headPtr );
          break;
default:
          printf( "Invalid choice.\n\n" );
          instructions();
          break;
      }

      printf( "? " );
      scanf( "%d", &choice );
   }
   printf( "End of run.\n" );
   return 0;
}
```

```
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value )
{
   QueueNodePtr newPtr;

   newPtr = malloc( sizeof( QueueNode ) );

   if ( newPtr != NULL ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      if ( isEmpty( *headPtr ) )
         *headPtr = newPtr;
      else
         ( *tailPtr )->nextPtr = newPtr;

      *tailPtr = newPtr;
   }
   else
      printf( "%c not inserted. No memory available.\n", value );
   return ;
}
```

```
char dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
{
   char value;
   QueueNodePtr tempPtr;

   value = ( *headPtr )->data;
   tempPtr = *headPtr;
   *headPtr = ( *headPtr )->nextPtr;

   if ( *headPtr == NULL )
      *tailPtr = NULL;

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

```c
int isEmpty( QueueNodePtr headPtr )
{
   return headPtr == NULL;
}

void printQueue( QueueNodePtr 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->nextPtr;
      }

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

# Όνομα συνάρτησης

- Οι παρενθέσεις ως τελεστής
  - Δήλωση/πρότυπο συνάρτησης

```
int f (int );
```

  - Κλήση συνάρτησης

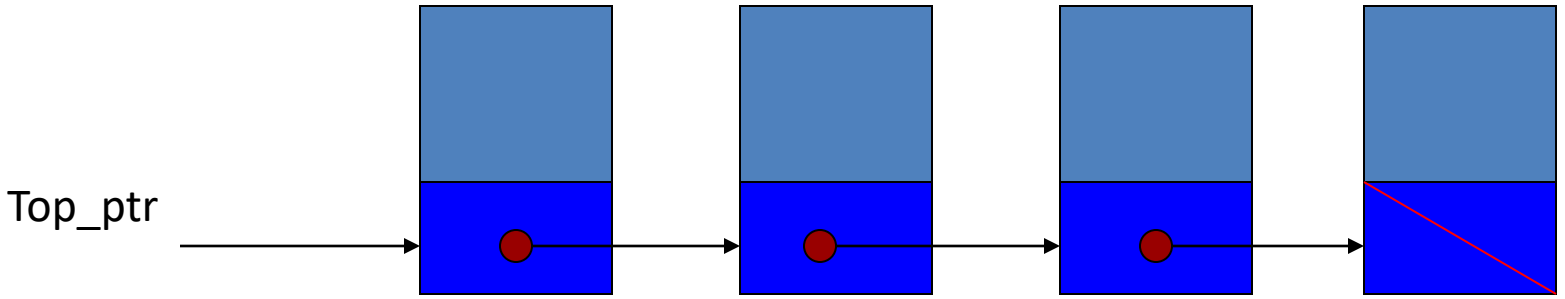
```
int a ;
a = f(5) ;
```

- Το όνομα συνάρτησης μόνο του $\Rightarrow$ διεύθυνση
- Μπορώ να δηλώσω σχετική **μεταβλητή**

# Πίνακας Δεικτών σε Συνάρτηση - Αρχικοποίηση

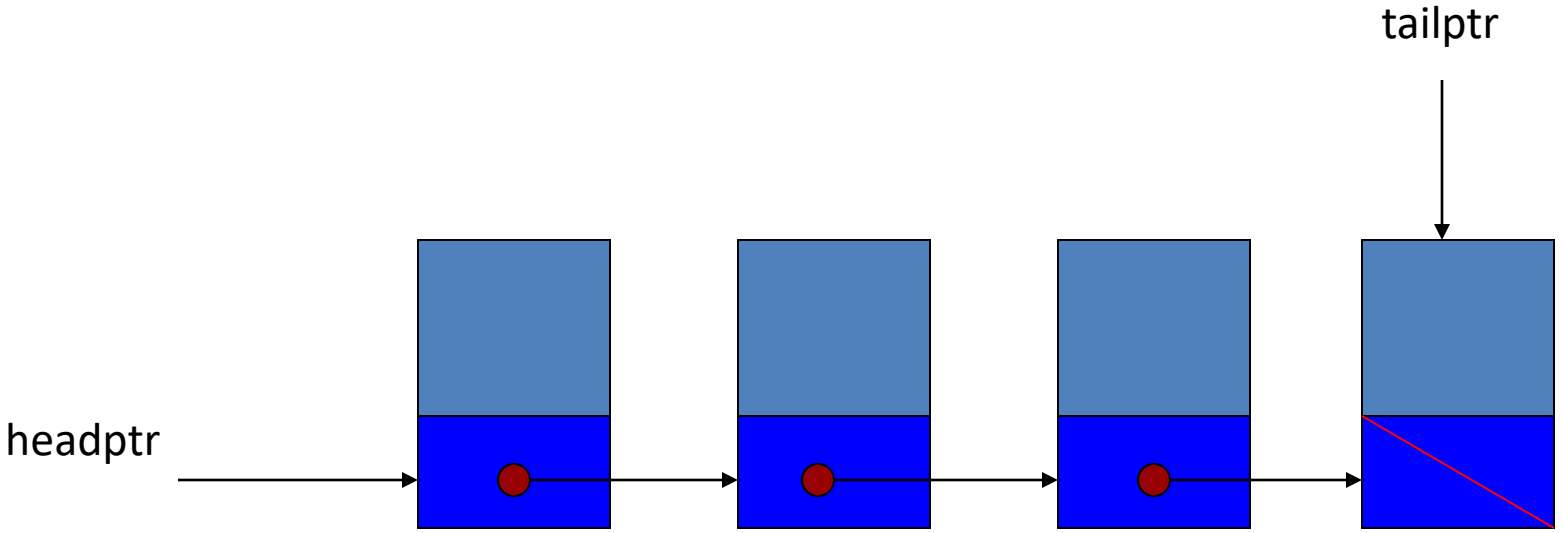
```
#include <stdio.h>
double one( double );
double two( double );

int main ( void ) {
        double (*f[2])(double) = {one, two} ;
        double x = 3.47, y;
        int i ;
        do {
                printf("select function:\t");
                scanf("%d", &i);
                y = (*f[i-1])( x) ;
                printf("result %g\n", y);
        } while (1) ;
        return 0;
}
double one(double x) { return (x + 1.0) ; }
double two(double x ) { return (x + 2.0) ; }
```

# Στοίβες - stacks



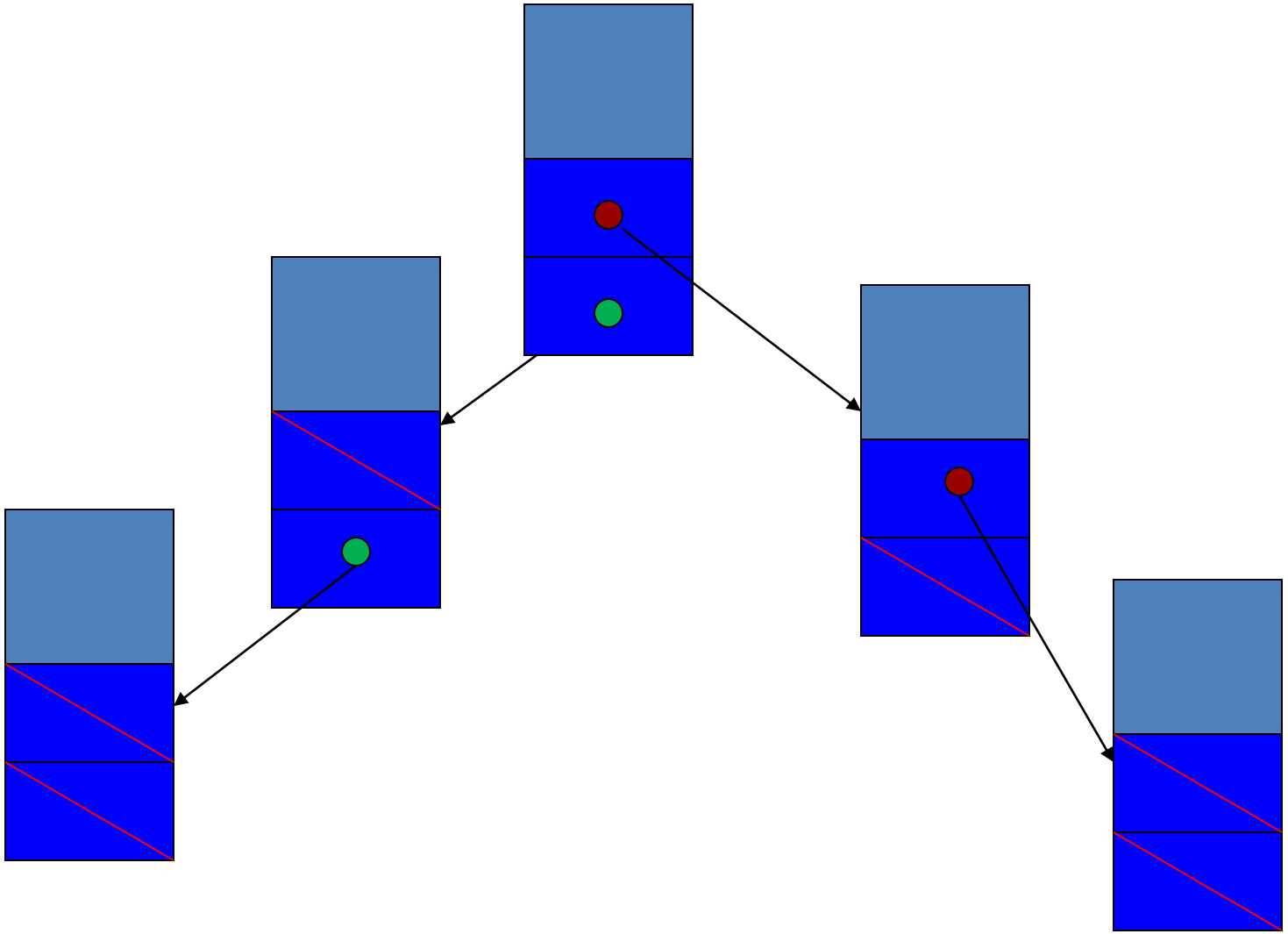

- Λειτουργίες push/pop

# Ουρές - queues



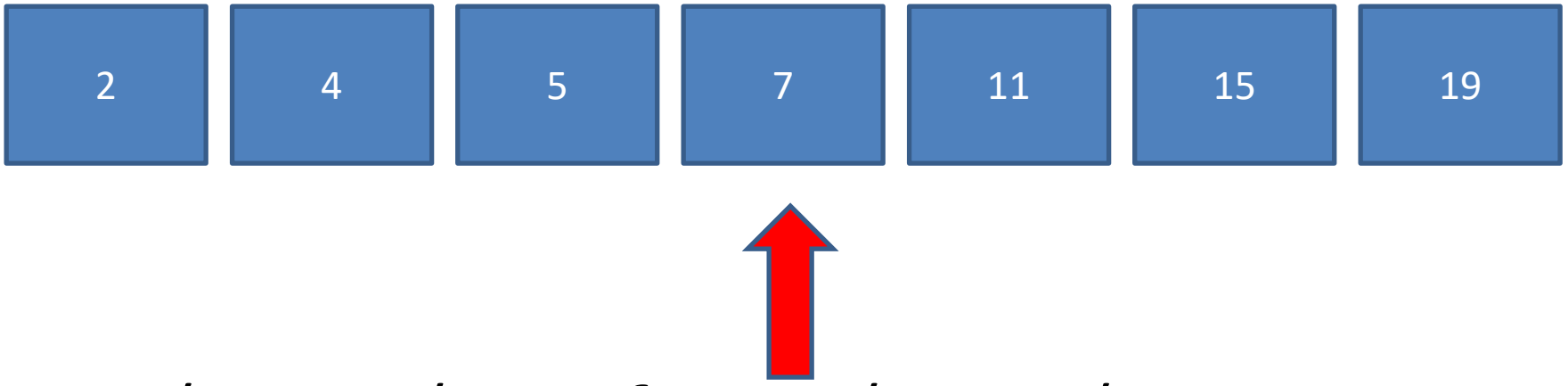

- Λειτουργίες enqueue/dequeue

# Δυαδικά Δένδρα



- Πρακτικές αναδρομικές συναρτήσεις
  - για κατασκευή του δυαδικού δένδρου και
  - για αναζήτηση

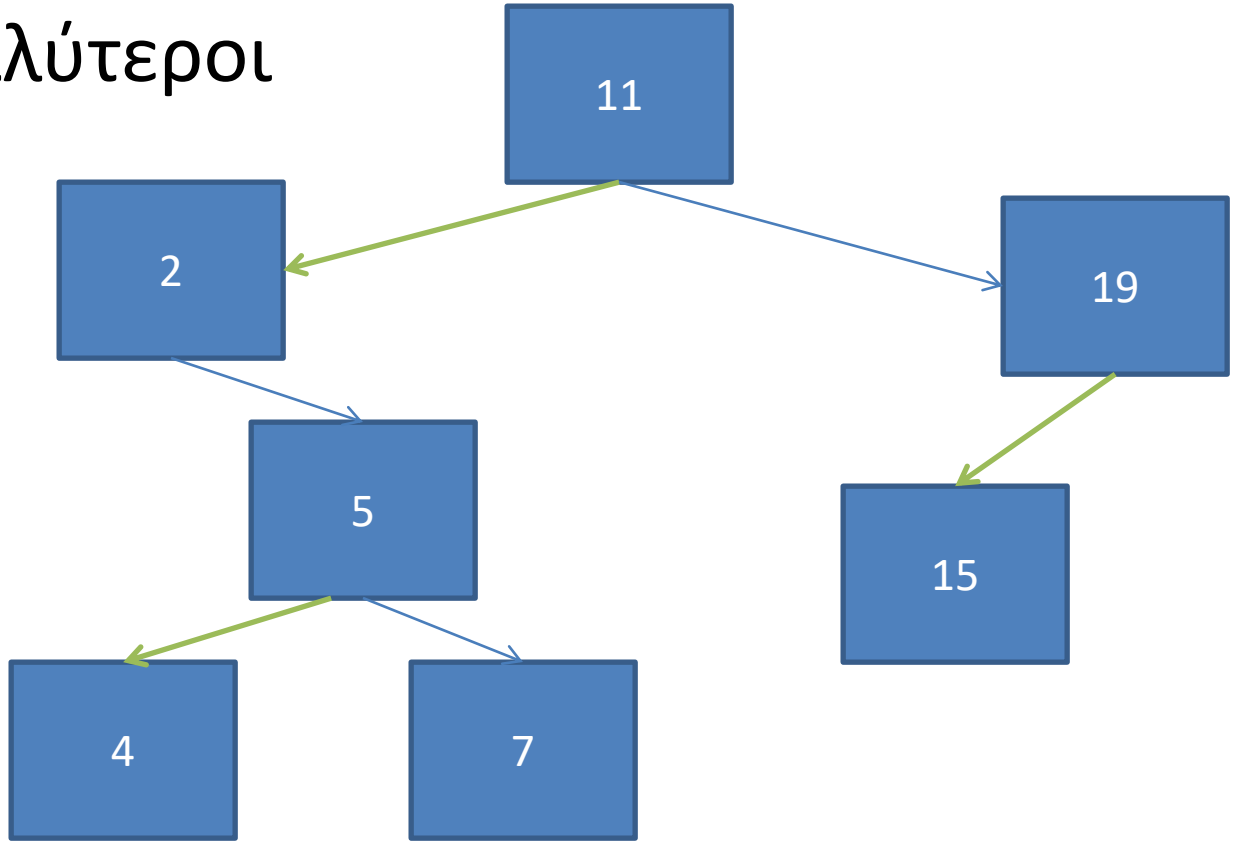
- Γραμμική δυαδική αναζήτηση σε πίνακα
  - Βρες αν υπάρχει το 5




- Πρέπει να είναι ταξινομημένος ο πίνακας
- Πώς προσθέτω στοιχεία σε ταξινομημένο πίνακα;

- Κατά την κατασκευή του δένδρου τοποθετούμε κόμβους σε θέση κατάλληλη.
- Το δένδρο μπορεί να επεκταθεί οποιαδήποτε στιγμή

- Παράδειγμα:  11, 2, 5, 19, 7, 15, 4
- Κανόνας: Αριστερά οι μικρότεροι, δεξιά οι μεγαλύτεροι



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

struct treeNode {
   struct treeNode *leftPtr;
   int data;
   struct treeNode *rightPtr;
};

typedef struct treeNode TreeNode;
typedef TreeNode *TreeNodePtr;

void insertNode( TreeNodePtr *, int );
void inOrder( TreeNodePtr );
void preOrder( TreeNodePtr );
void postOrder( TreeNodePtr );
```

```c
int main()
{
   int i, item;
   TreeNodePtr rootPtr = NULL;

   srand( time( NULL ) );

   /* insert random values between 1 and 15 in the tree */
   printf( "The numbers being placed in the tree are:\n" );

   for ( i = 1; i <= 10; i++ ) {
      item = rand() % 15;
      printf( "%3d", item );
      insertNode( &rootPtr, item );
   }

   /* traverse the tree preOrder */
   printf( "\n\nThe preOrder traversal is:\n" );
   preOrder( rootPtr );

   /* traverse the tree inOrder */
   printf( "\n\nThe inOrder traversal is:\n" );
   inOrder( rootPtr );

   /* traverse the tree postOrder */
   printf( "\n\nThe postOrder traversal is:\n" );
   postOrder( rootPtr );

   return 0;
}
```

# Τοποθέτηση

- Κενή θέση;
  - Αν ναι, συνέδεσε τον κόμβο και τέλος.
  - Αν όχι,
    - Αν η τιμή που θέλω να τοποθετήσω είναι μικρότερη από την τρέχουσα,  τοποθέτησε  αριστερά
    - Αν είναι μεγαλύτερη, τοποθέτησε δεξιά
    - Αν είναι ίση, η τιμή υπάρχει στο δένδρο, τέλος.

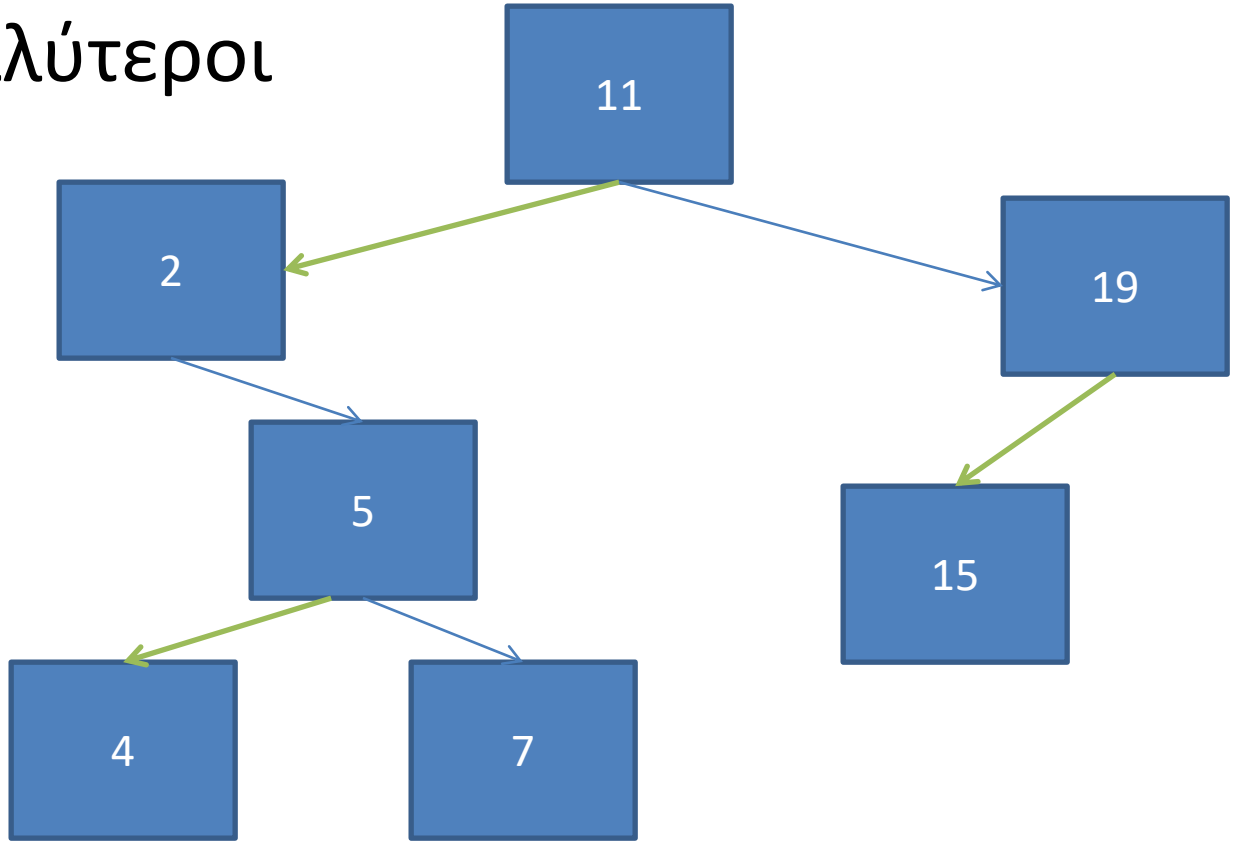
```
void insertNode( TreeNodePtr *treePtr, int value )
{
   if ( *treePtr == NULL ) {    /* *treePtr is NULL */
      *treePtr = malloc( sizeof(TreeNode) );

      if ( *treePtr != NULL ) {
         ( *treePtr )->data = value;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      }
      else
         printf( "%d not inserted. No memory available.\n",
                 value );
   }
   else
      if ( value < ( *treePtr )->data )
         insertNode( &( ( *treePtr )->leftPtr ), value );
      else if ( value > ( *treePtr )->data )
         insertNode( &( ( *treePtr )->rightPtr ), value );
      else
         printf( "dup" );
}
```

```
void inOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      inOrder( treePtr->leftPtr );
      printf( "%3d", treePtr->data );
      inOrder( treePtr->rightPtr );
   }
}
```

```
void preOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      printf( "%3d", treePtr->data );
      preOrder( treePtr->leftPtr );
      preOrder( treePtr->rightPtr );
   }
}

void postOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      postOrder( treePtr->leftPtr );
      postOrder( treePtr->rightPtr );
      printf( "%3d", treePtr->data );
   }
}
```

- Πρακτικές αναδρομικές συναρτήσεις
  - για κατασκευή του δυαδικού δένδρου και
  - για αναζήτηση

- Παράδειγμα:  11, 2, 5, 19, 7, 15, 4
- Κανόνας: Αριστερά οι μικρότεροι, δεξιά οι μεγαλύτεροι



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

struct treeNode {
   struct treeNode *leftPtr;
   int data;
   struct treeNode *rightPtr;
};

typedef struct treeNode TreeNode;
typedef TreeNode *TreeNodePtr;

void insertNode( TreeNodePtr *, int );
void inOrder( TreeNodePtr );
void preOrder( TreeNodePtr );
void postOrder( TreeNodePtr );
```

```c
int main()
{
   int i, item;
   TreeNodePtr rootPtr = NULL;

   srand( time( NULL ) );

   /* insert random values between 1 and 15 in the tree */
   printf( "The numbers being placed in the tree are:\n" );

   for ( i = 1; i <= 10; i++ ) {
      item = rand() % 15;
      printf( "%3d", item );
      insertNode( &rootPtr, item );
   }

   /* traverse the tree preOrder */
   printf( "\n\nThe preOrder traversal is:\n" );
   preOrder( rootPtr );

   /* traverse the tree inOrder */
   printf( "\n\nThe inOrder traversal is:\n" );
   inOrder( rootPtr );

   /* traverse the tree postOrder */
   printf( "\n\nThe postOrder traversal is:\n" );
   postOrder( rootPtr );

   return 0;
}
```

# Τοποθέτηση

- Κενή θέση;
  - Αν ναι, συνέδεσε τον κόμβο και τέλος.
  - Αν όχι,
    - Αν η τιμή που θέλω να τοποθετήσω είναι μικρότερη από την τρέχουσα,  τοποθέτησε  αριστερά
    - Αν είναι μεγαλύτερη, τοποθέτησε δεξιά
    - Αν είναι ίση, η τιμή υπάρχει στο δένδρο, τέλος.

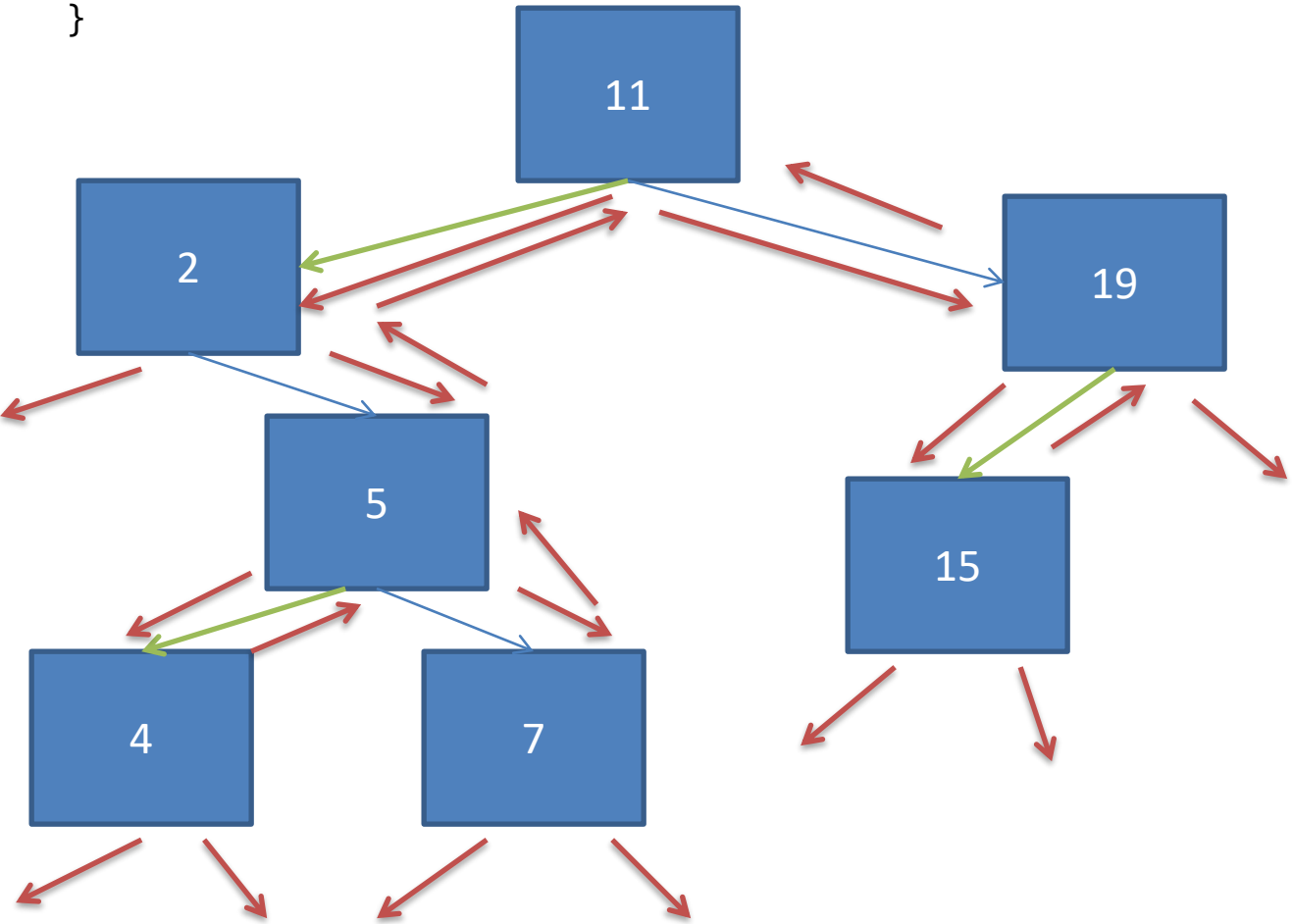
```
void insertNode( TreeNodePtr *treePtr, int value )
{
   if ( *treePtr == NULL ) {    /* *treePtr is NULL */
      *treePtr = malloc( sizeof(TreeNode) );

      if ( *treePtr != NULL ) {
         ( *treePtr )->data = value;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      }
      else
         printf( "%d not inserted. No memory available.\n",
                 value );
   }
   else
      if ( value < ( *treePtr )->data )
         insertNode( &( ( *treePtr )->leftPtr ), value );
      else if ( value > ( *treePtr )->data )
         insertNode( &( ( *treePtr )->rightPtr ), value );
      else
         printf( "dup" );
}
```

```
void inOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      inOrder( treePtr->leftPtr );
      printf( "%3d", treePtr->data );
      inOrder( treePtr->rightPtr );
   }
}
```

```
void preOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      printf( "%3d", treePtr->data );
      preOrder( treePtr->leftPtr );
      preOrder( treePtr->rightPtr );
   }
}

void postOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      postOrder( treePtr->leftPtr );
      postOrder( treePtr->rightPtr );
      printf( "%3d", treePtr->data );
   }
}
```

```
void inOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      inOrder( treePtr->leftPtr );
      printf( "%3d", treePtr->data );
      inOrder( treePtr->rightPtr );
   }
}
```

# inOrder traversal




2

4

5

7

11

15

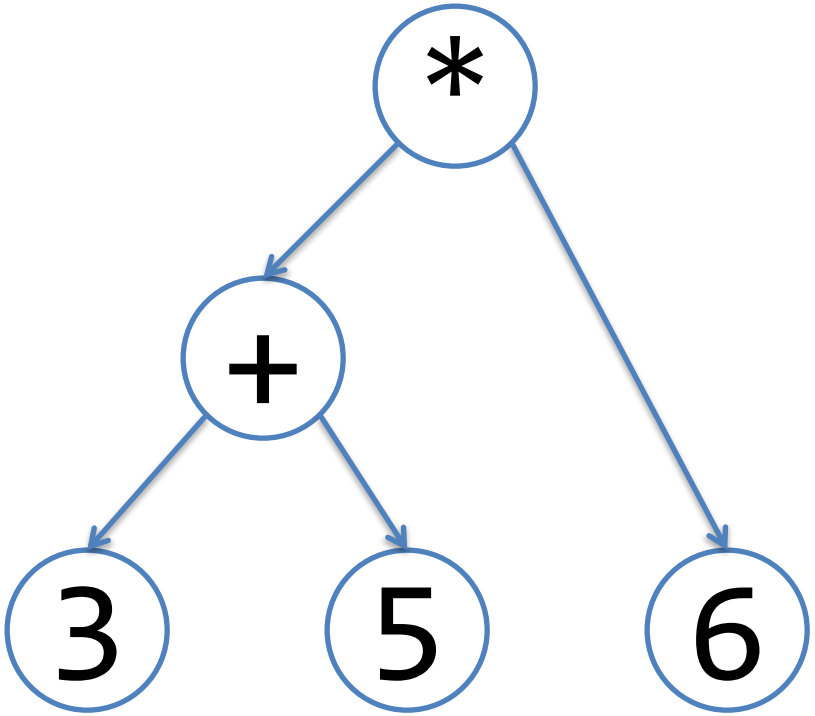
19

```
void preOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      printf( "%3d", treePtr->data );
      preOrder( treePtr->leftPtr );
      preOrder( treePtr->rightPtr );
   }
}
```

# Παραδείγματα pre-order traversal

- Αντιγραφή δένδρου
  - Για κάθε κόμβο
    - Αντιγράφει τον κόμβο
    - Αντιγράφει αριστερό υπο-δένδρο
    - Αντιγράφει δεξί υπο-δένδρο
- Εξαγωγή αναπαράστασης προθέματος (prefix)

# *Pre-order* traversal



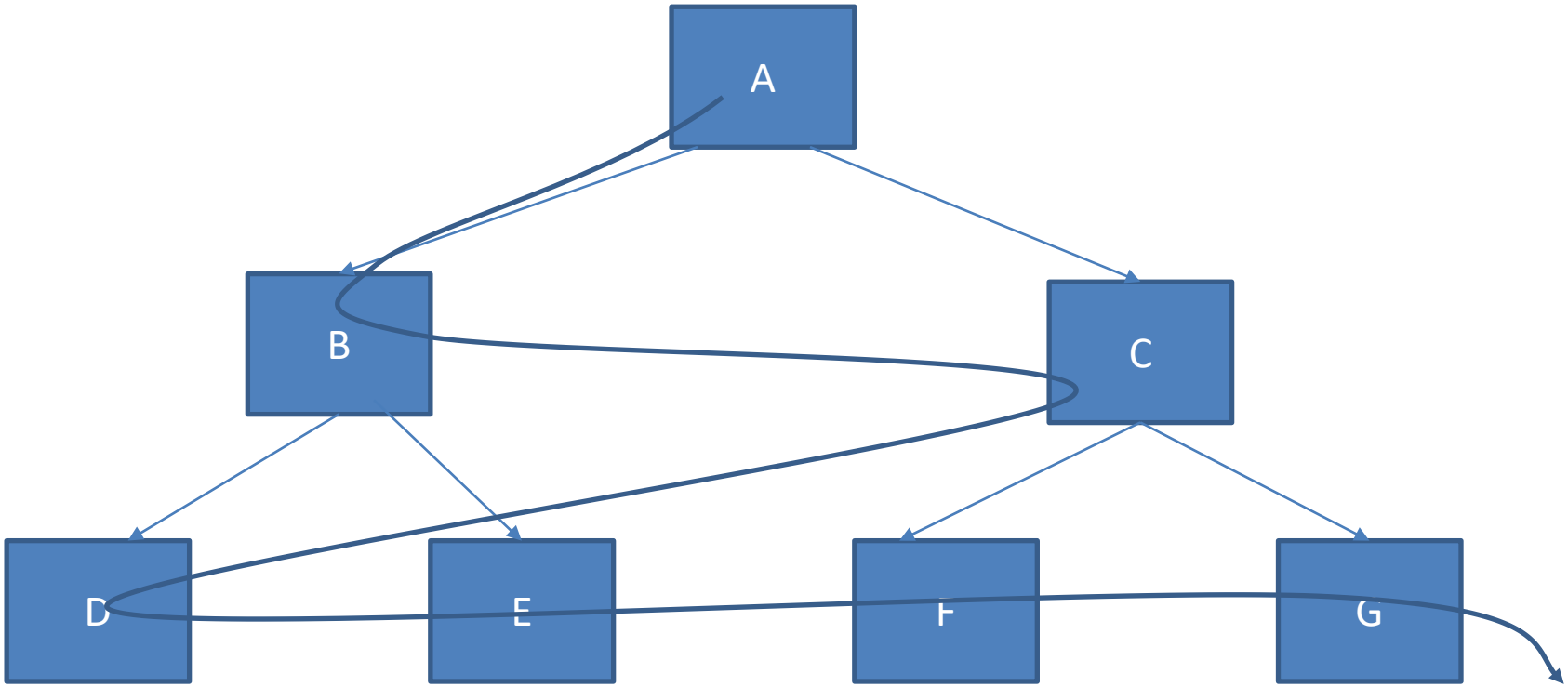

(3 + 5) * 6

* + 3 5 6

```
void postOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      postOrder( treePtr->leftPtr );
      postOrder( treePtr->rightPtr );
      printf( "%3d", treePtr->data );
   }
}
```

# Παραδείγματα post-order traversal

- Διαγραφή δένδρου
  - Για κάθε κόμβο
    - Διαγράφει αριστερό υπο-δένδρο
    - Διαγράφει  δεξί υπο-δένδρο
    - Διαγράφει κόμβο
- Εξαγωγή αναπαράστασης postfix




A

B C

C D E

D E F G

E F G

F G

G

# Αναζήτηση κατά πλάτος
# Breadth-First Search

Φτιάξε μια άδεια ουρά
Τοποθέτηση τη ρίζα του δένδρου στην ουρά
Όσο η ουρά δεν είναι άδεια:
    Πάρε ένα κόμβο από την ουρά
    Τοποθέτησε όλους τους απογόνους στην ουρά
    Τύπωσε τον κόμβο

```
void BFS(TreeNodePtr treePtr) {

        QueueNodePtr headPtr = NULL, tailPtr = NULL;

        TreeNodePtr temp;

        enqueue(&headPtr, &tailPtr, treePtr);

        while (!isEmpty(headPtr)) {
                temp = dequeue(&headPtr, &tailPtr);
                if (temp->leftPtr != NULL)
                        enqueue(&headPtr, &tailPtr, temp->leftPtr);
                if (temp->rightPtr != NULL)
                        enqueue(&headPtr, &tailPtr, temp->rightPtr);
                printf("%3d", temp->data);
        }
        return;
}
```

ΠΡΟΣΟΧΗ: ΕΝΔΕΙΚΤΙΚΟΣ ΚΩΔΙΚΑΣ

ΜΗ ΧΡΗΣΙΜΟΠΟΙΗΣΕΤΕ ΑΥΤΌΝ ΓΙΑ ΤΑ ΠΑΡΑΔΟΤΕΑ!

ΝΑ ΑΝΑΠΤΥΞΕΤΕ ΔΙΚΕΣ ΣΑΣ ΛΥΣΕΙΣ!

# ΕΡΓΑΣΤΗΡΙΟ ΠΕΝΤΕ

# Λογικοί τελεστές

- `int a;`
- `a = (έκφραση1) && (έκφραση2);  /* AND */`
- `a = (έκφραση2) || (έκφραση2);  /* OR  */`
- `a = !(έκφραση);  /* NOT */`

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

int and (int, int);
int not (int);
int or(int, int);

typedef struct gate {
        int (*type)();
        struct gate *in1;
        struct gate *in2;
        int result;
        } Gate;
```

Χρησιμοποιώ αναδρομή.

Δεν αποθηκεύω αναλυτικά την τιμή της εξόδου.

Η κάθε πύλη ρωτάει τις προηγούμενες πύλες για να μάθει τις τιμές των εξόδων τους.

ΜΗΝ ΧΡΗΣΙΜΟΠΟΙΗΣΕΤΕ ΑΥΤΌΝ ΤΟΝ ΚΩΔΙΚΑ ΩΣ ΕΧΕΙ!!!

# ΜΙΑ ΛΥΣΗ

```c
#include <stdio.h>
#include <stdlib.h>
int and (int, int);
int not (int);
int or(int, int);
int input();
typedef struct gate {
        int (*type)();
        struct gate * in1;
        struct gate * in2;
        } Gate;
int evaluate(Gate);
```

```c
int main(void) {
  int d;
  Gate input1 = { input, NULL, NULL};
  Gate input2 = { input, NULL, NULL};
  Gate input3 = { input, NULL, NULL};
  Gate g1 = {and, &input1, &input2};
  Gate g2 = { or, &g1, &input3};
  Gate g3 = {not, &g2, NULL};
  d = evaluate(g3);
  printf("result: %d\n", d);
  system("PAUSE");
  return 0;
}
```

```c
int and (int a, int b) {
    return a * b;
}

int or (int a, int b) {
    return a+b>0;
}
int not (int a) {
    return 1 -a ;
}

int input () {
    int a;
    printf("Enter input: ");
    scanf("%d", &a);
    return a;
}
```

```
int evaluate(Gate g) {
    int out;
    int a1=-1, a2=-1;

    if (g.in1!=NULL)
       a1 = evaluate(*g.in1);
    if (g.in2!=NULL)
       a2 = evaluate(*g.in2);

    out = (*g.type)(a1,a2);

 return out;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#define AND 0
#define OR 1
#define NOT 2
#define INPUT 3

int and (int, int);
int not (int);
int or (int, int);
int input();

typedef struct gate {
        int (*type)();
        struct gate * in1;
        struct gate * in2;
        } Gate;

typedef struct gate4file {
                int type;
                int in1;
                int in2;
} Gate4file;
```

```
void file2eval ( Gate g[], const Gate4file gf[], int gates)
{
        int i;

        int (*f[])()= {and, or, not, input};

        for (i=0;i<gates; i++) {
                g[i].type = f[gf[i].type];
                g[i].in1 = gf[i].in1!=-1?&g[gf[i].in1]:NULL;
                g[i].in2 = gf[i].in2!=-1?&g[gf[i].in2]:NULL;
        }

}
```

```
int main( ) {
  int d;
  FILE *f;

  Gate g[6];
  Gate h[6];
  g[0].type= input; g[0].in1= NULL; g[0].in2= NULL;     /* input 1 */
  g[1].type= input; g[1].in1= NULL; g[1].in2= NULL;     /* input 2 */
  g[2].type= input; g[2].in1= NULL; g[2].in2= NULL;     /* input 3 */
  g[3].type= and;   g[3].in1= &g[0];g[3].in2= &g[1];    /* g1 */
  g[4].type= or;   g[4].in1= &g[3];g[4].in2= &g[2];     /* g2 */
  g[5].type= not;  g[5].in1= &g[4];g[5].in2= NULL;      /* g3 */

  Gate4file gf[6] = { {INPUT, -1, -1},
                      {INPUT, -1, -1},
                      {INPUT, -1, -1},
                      {AND,  0,  1},
                      {OR,  3,  2},
                      {NOT,  4, -1}};
  Gate4file hf[6];
  f = fopen("circ2.dat", "wb");
  fwrite(gf, sizeof(Gate4file), 6, f);
  fclose(f);
  f = fopen("circ2.dat", "rb");
  fread(hf, sizeof(Gate4file), 6, f);
  fclose(f);
  file2eval(h, hf, 6);
  d = evaluate(h[5]);
  printf("result: %d\n", d);
  return 0;
}
```