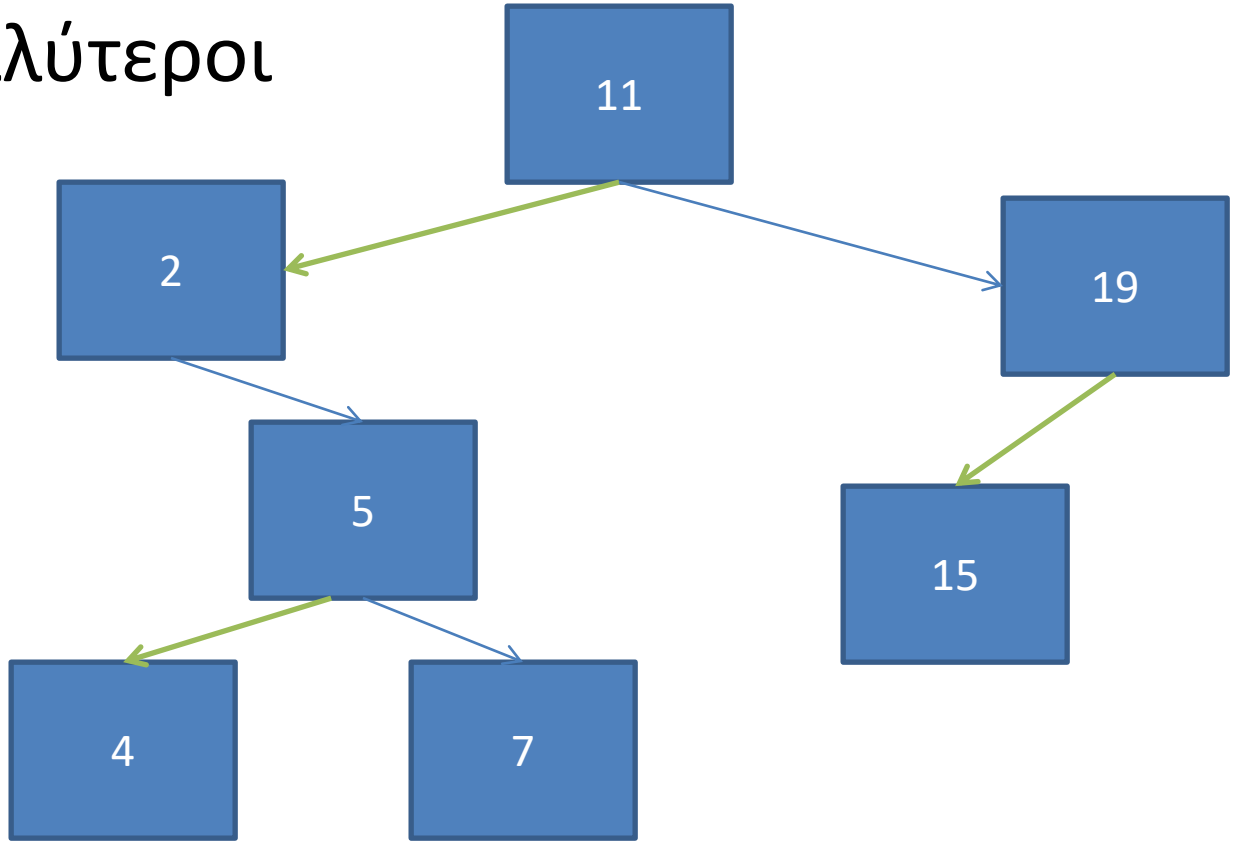
Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

- 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

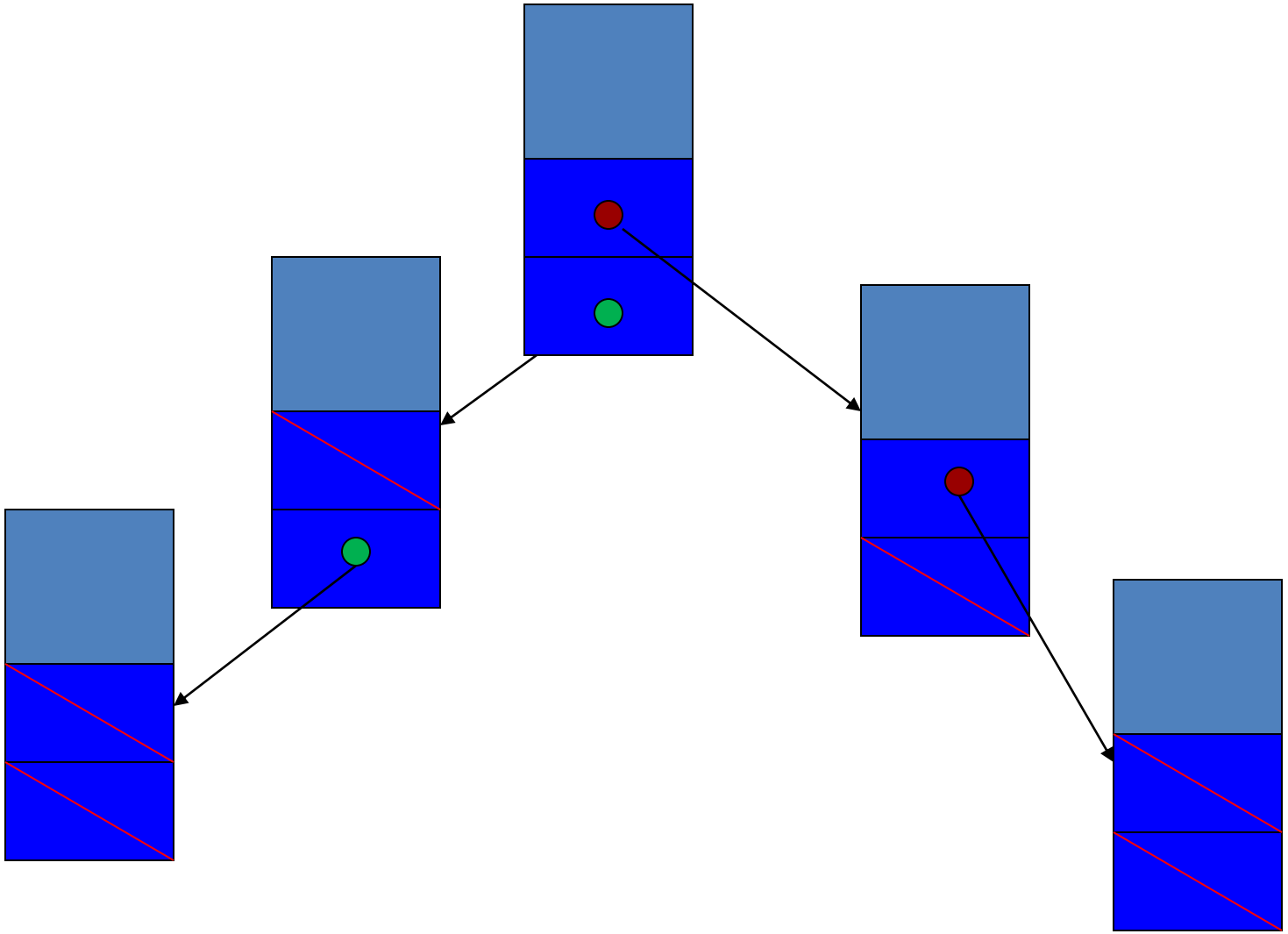
- Κενή θέση;
  - Αν ναι, συνέδεσε τον κόμβο και τέλος.
  - Αν όχι,
    - Αν η τιμή που θέλω να τοποθετήσω είναι μικρότερη από την τρέχουσα, τοποθέτησε αριστερά
    - Αν είναι μεγαλύτερη, τοποθέτησε δεξιά
    - Αν είναι ίση, η τιμή υπάρχει στο δένδρο, τέλος.

```
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 );
   }
}
```

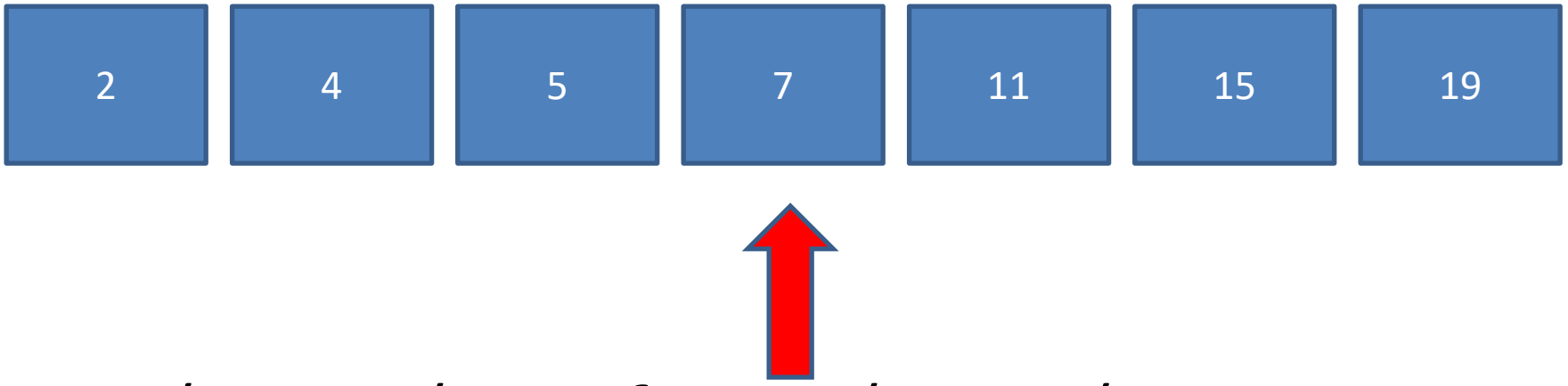
```
void preOrder( TreeNodePtr treePtr )
{
   if ( treePtr != NULL ) {
      printf( "%3d", treePtr->data );
      preOrder( treePtr->leftPtr );
      preOrder( treePtr->rightPtr );
   }
}

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

Δυαδικά Δένδρα



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

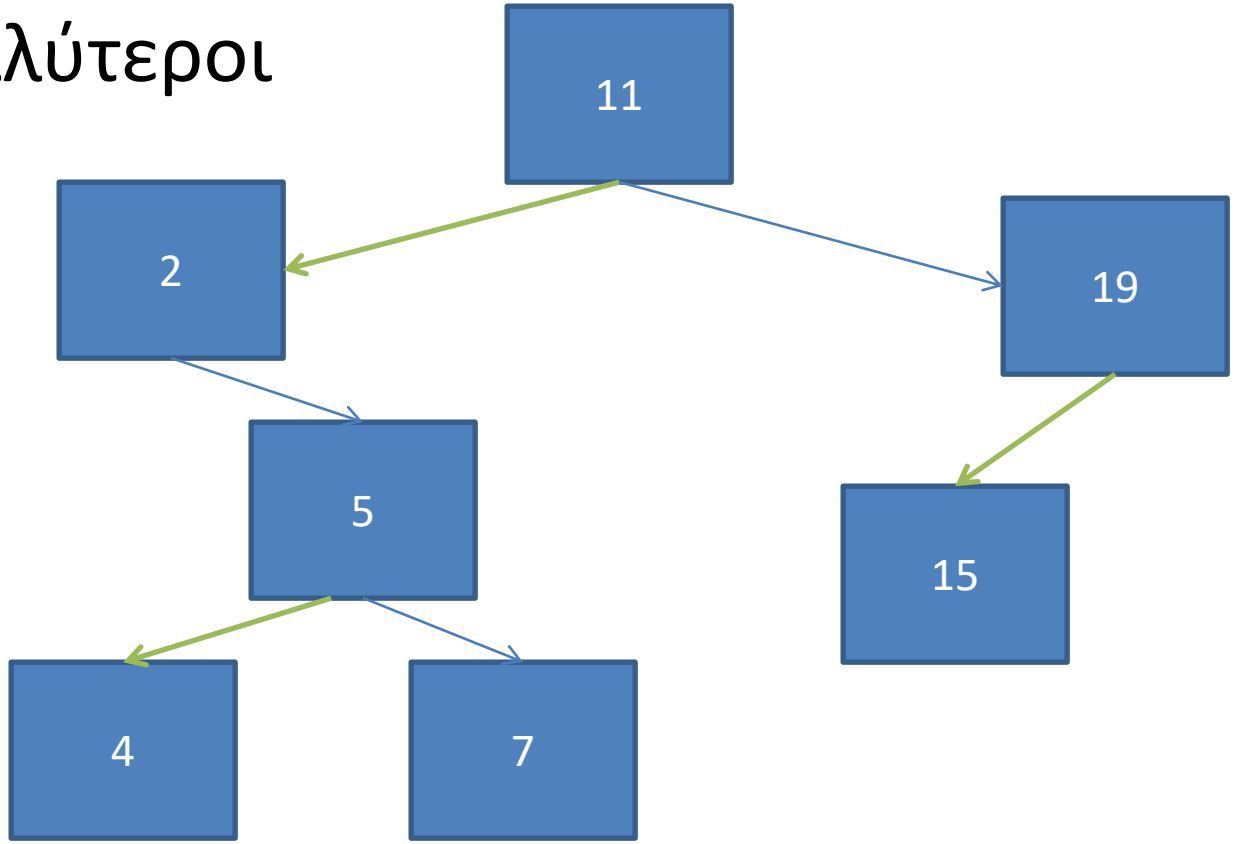
- Γραμμική δυαδική αναζήτηση σε πίνακα
  - Βρες αν υπάρχει το 5



- 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.

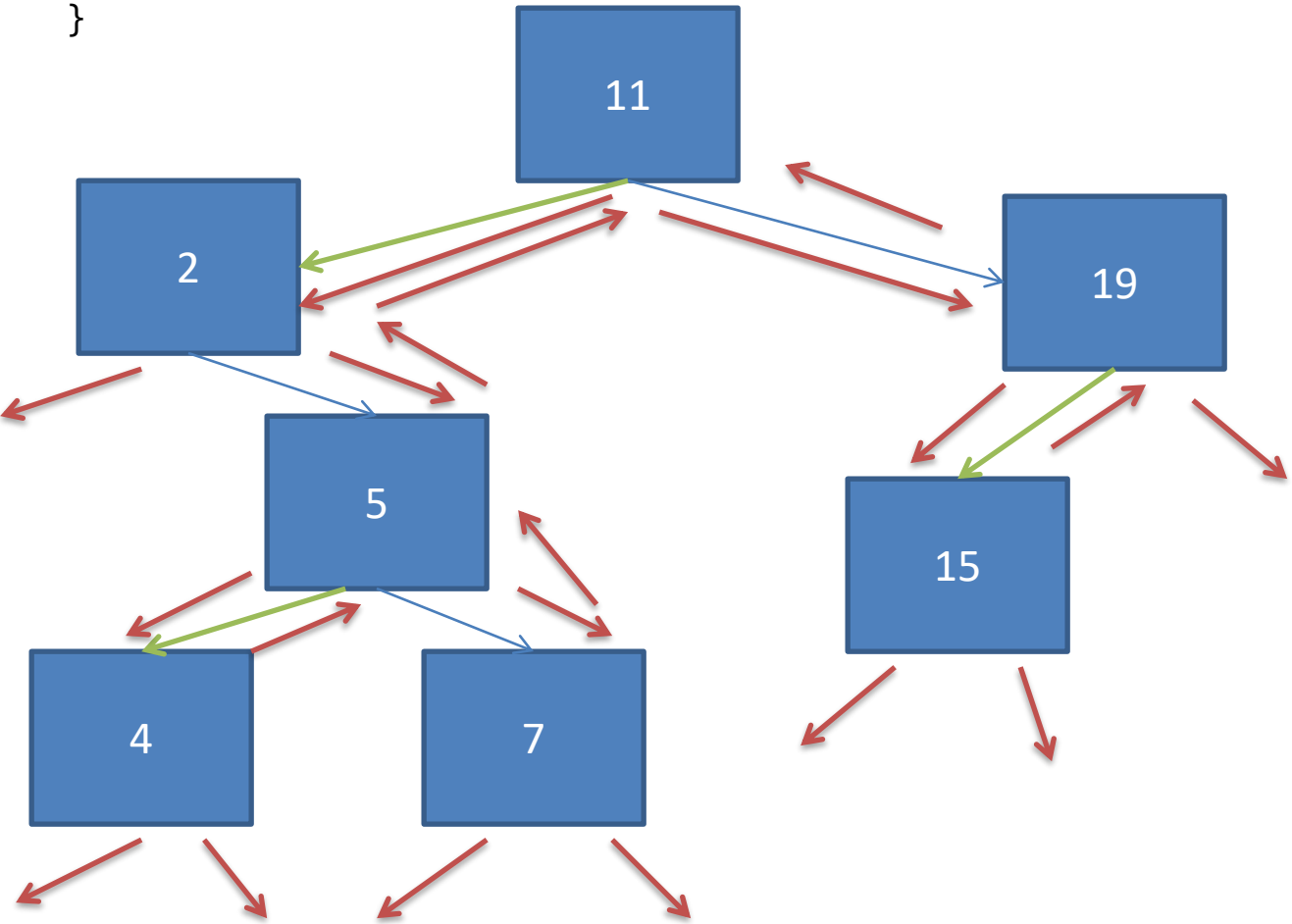
```
void insertNode( TreeNodePtr *treePtr, int value )
{
   if ( *treePtr == NULL ) {    /* *treePtr is NULL */
      *treePtr = malloc( sizeof(TreeNode) );

      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 );
   }
}
```

```
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

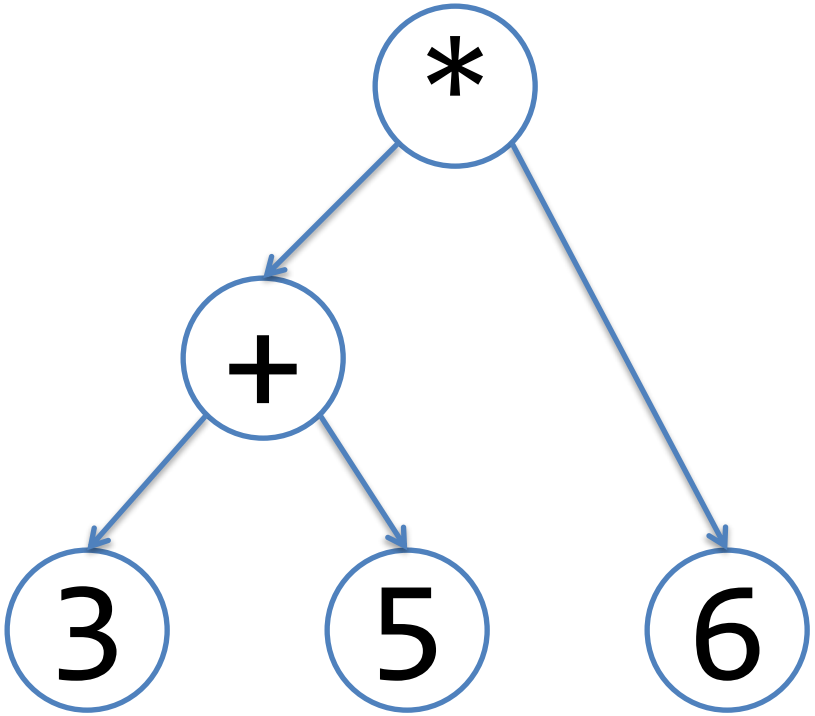
19

```
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

qsort/bsearch

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

int compareints(const void *, const void *);
void report (int *, int);

int main (void)
{
  int * pItem;
  int data[] = { 45, 2, 60, 34, 12, 25 };
  int len ;
  int key = 45;

  len = sizeof data/sizeof data[0];

  report(data, len);

  qsort (data, len, sizeof (int), compareints);

  report(data, len);

  pItem = (int *) bsearch (&key, data, len, sizeof (int), compareints);


  if (pItem!=NULL)
    printf ("%d found in data.\n",*pItem);
  else
    printf ("Cannot find %d.\n",key);

  return EXIT_SUCCESS;
}
```

```c
int compareints (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}

void report(int *x , int len) {
  int i;
  for (i=0; i<len; i++) {
     printf("%d: %d\n", i, x[i]);
   }
   return ;
}
```

```c
int main (void) {
  int len;
  Item items[] ={ {10, "a"}, {9, "m"}, {15, "d"}, {20,"hello"}, {1, "f"}};
  Item * sItem ;
  Item keyItem ;

  len = sizeof items/sizeof items[0];

  /* sort for x */
  reportitems(items, len);
  qsort(items, len, sizeof (Item), compareitems);

  reportitems(items, len);

  /* sort for name */
  qsort(items, len, sizeof (Item), comparestrs);

  reportitems(items, len);

  /* search for an item that contains "hello" */
  strcpy(keyItem.name, "hello");
  sItem = bsearch(&keyItem, items, len , sizeof (Item), comparestrs);

  if (sItem != NULL) { /* if found, print all members of item */
          printf("found:\n\t");
          reportitems(sItem, 1); /* print found item */
  }
  else
          printf("%s not found!\n", keyItem.name);

  return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct item {
              int x;
              char name[10];
} Item;

void reportitems(Item *, int );
int compareitems(const void *, const void *);
int comparestrs(const void *, const void *);
```

Κανόνας για τάξινόμηση/αναζήτηση:
int (*f)(const void *, const void *)

```
int compareitems(const void * a, const void * b) {
        return ( ((Item *) a)->x - ((Item *) b)->x);
}

int comparestrs(const void *a, const void *b) {
        return strcmp( ((Item *) a)->name, ((Item *)b)->name );
}


void reportitems(Item * it, int len) {
        int i;
        for (i=0;i<len;i++) {
                printf("%2d:%s\n", it[i].x, it[i].name);
        }
        return ;
}
```

#define

```c
#include <stdio.h>
#define N 5

int main( ) {

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

  return 0;
}
```

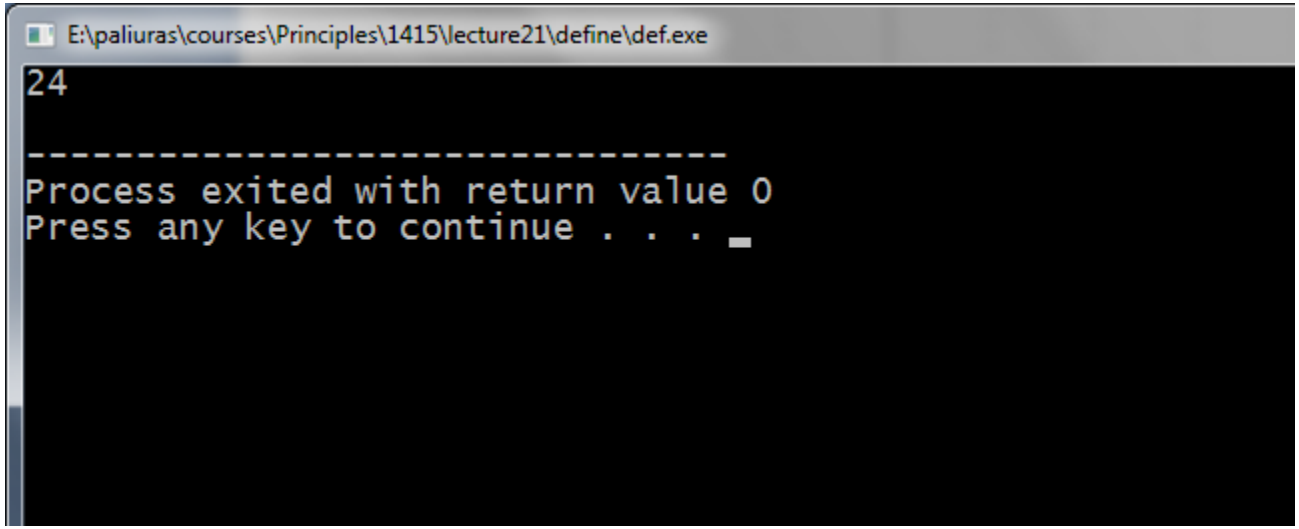
Preprocessor macros: Σωστό ;;;

```c
#include <stdio.h>
#define MYSQUARE(X)  X * X

int main( ) {

  printf("%d\n", MYSQUARE(4));

  return 0;
}
```

«Λάθος»...γιατί;

```c
#include <stdio.h>
#define MYSQUARE(X)  X * X

int main( ) {

        printf("%d\n", MYSQUARE(4+4));

        return 0;
}
```

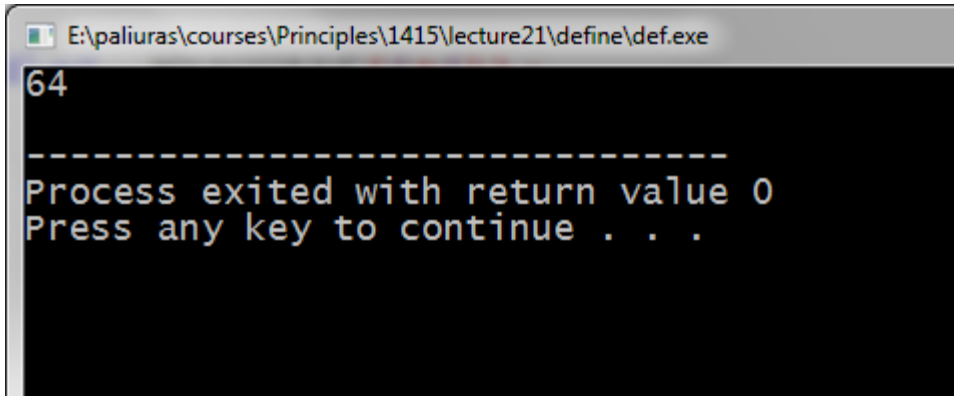
4+4*4+4



Preprocessor macros

```
#include <stdio.h>
#define MYSQUARE(X)  (X) * (X)

int main( ) {

  printf("%d\n", MYSQUARE(4+4));

  return 0;
}
```

(4+4)*(4+4)



Τύπος ως παράμετρος;;;

```
#include <stdio.h>
#include <stdlib.h>
#define MYSQUARE(X)  (X) * (X)
#define reportsize(x) sizeof(x)

int main( ) {

      printf("%d\n", MYSQUARE(4+4));
      printf("%d\n", reportsize(int));

      return 0;
}
```

φαινομενικά

Variadic functions

```
#include <stdarg.h>

int findmax(int, ...);
```

Τελεστής έλλειψης

```
int findmax( int n, ...) {
        int i;
        int mmax, temp;
        va_list args;
        va_start(args, n);
        mmax = va_arg(args, int);
        /* for loop executed when n>1 only */
        for (i=2; i<=n; i++) {
                temp = va_arg(args, int);
                if (temp>mmax) mmax= temp;
        }

        va_end(args);
        return mmax;
}
```

Macro va_arg( )

```c
int main ()
{

   printf("%d\n", findmax(3, 1, 2, 3));
   printf("%d\n", findmax(5, 1, 2, 13, 4, 7));

   return 0;
}
```

union

```c
#include <stdio.h>

typedef union t {
          int x;
          char bytes[4] ;
         } T ;


int main( void ) {
   int i ;
   T a;
   a.x = 0x00000a0a;

   printf("size: %d bytes\n", sizeof (T));

   for (i=0; i<4; i++) {
        printf("%2X\n", * (unsigned char *) & a.bytes[i]);
   }

   return 0;

}
```

```c
#include <stdio.h>
typedef union d {
            int xint;
            double xdouble;
            float xfloat;
} X ;
typedef enum {INT, DOUBLE, FLOAT} Type;
typedef struct  {
               Type type;
               X x;
} Flextype;
void myprint(Flextype);
int main( ) {
            int i ;
            Flextype a;
            a.type = INT;
            a.x.xint = 4;
            myprint(a);

            a.type = FLOAT;
            a.x.xfloat = 3.5;

            myprint(a);
            return 0;
}
void myprint(Flextype x) {
            X data;
            data = x.x;
            switch (x.type) {
                        case INT:    printf("integer: %d\n", data.xint);
                                                          break;
                        case FLOAT: printf("float: %f\n", data.xfloat);
                                                          break;
                        case DOUBLE:printf("double: %g\n", data.xdouble);
                                                          break;
                        default:     printf ("unknown %d\n", x.type);
                                                          break;
            }
}
```