Διαδικαστικός προγραμματισμός

Βασίλης Παλιουράς

#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;

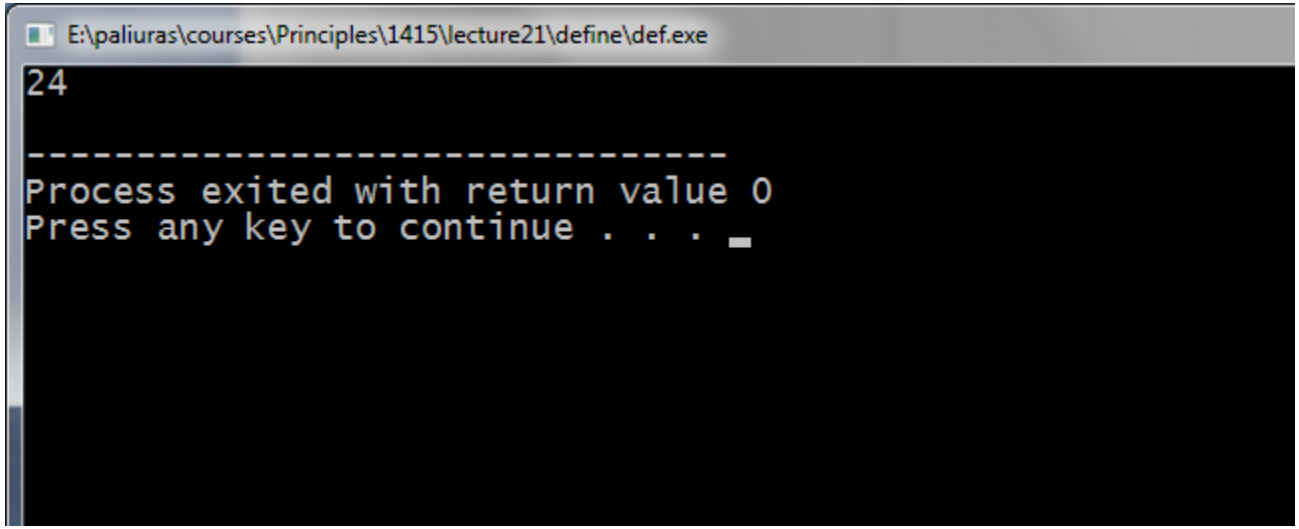
```c
#include <stdio.h>
#define MYSQUARE(X)  X * X

int main( ) {

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

        return 0;
}
```

4+4*4+4



Preprocessor macros

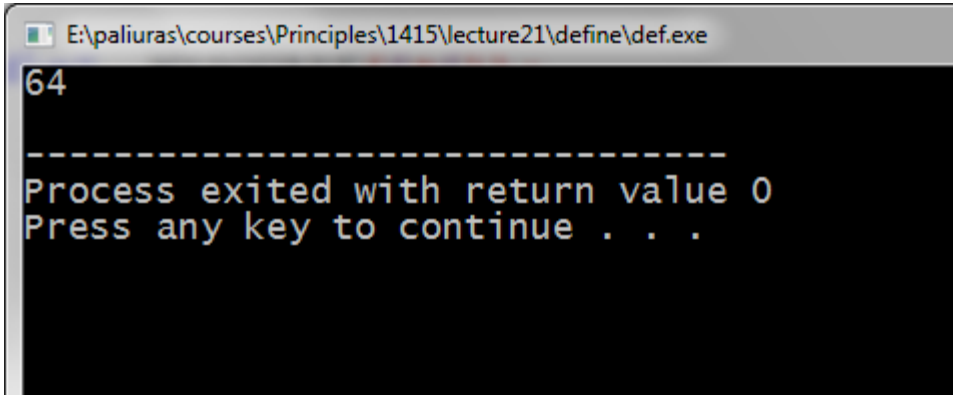
```
#include <stdio.h>
#define MYSQUARE(X)  (X) * (X)

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;
            }
}
```

- Δίνονται οι κάτωθι τύποι

```
typedef struct node {
  int data;
  struct node * next; } LstNode;

typedef LstNode * List;
typedef LstNode * ListNodePtr;
```

- Να γραφεί συνάρτηση που να τοποθετεί στοιχεία σε μία απλά διασυνδεμένη λίστα ώστε η λίστα να παραμένει ταξινομημένη κατά φθίνουσα σειρά

```
void createsortedlst(List *newlstptr, int NoOfElems ) {
        int i ;
        ListNodePtr tempPtr ;

        srand( time (0) ) ;

        for (i=0; i < NoOfElems; i++ ) {
                tempPtr = createnode(rand() % 10);
                insertsorted(newlstptr, tempPtr  );
        }

        return ;
}
```

```
void insertsorted(List * newlstptr, ListNodePtr tempPtr) {
        ListNodePtr iter , previous = NULL;

        if (*newlstptr == NULL)  {*newlstptr = tempPtr; return;}

        iter = *newlstptr;
        for (iter = *newlstptr; iter != NULL; iter = iter->next ) {
                if (iter->data < tempPtr ->data) {
                        if (previous == NULL) {
                                *newlstptr = tempPtr;
                                tempPtr->next = iter;
                                return;
                        }
                        previous->next = tempPtr;
                        tempPtr->next = iter;
                        return;
                }
                previous = iter;
        }
        previous->next = tempPtr;
        return ;
}
```

- Να γραφεί συνάρτηση που λαμβάνει ως όρισμα μια απλά διασυνδεδεμένη λίστα και διαχωρίζει τα στοιχεία της σε δύο άλλες λίστες, μια για τα περιττά και μια τα άρτια στοιχεία.

- Προσέξτε ότι ζητείται να μην δημιουργηθούν νέοι κόμβοι.

```
void breaklst(     List * mylstPtr, List * oddlstPtr,  List * evenlstPtr) {
        LstNode * temp;

        while (1)  {
                if (*mylstPtr == NULL) return ; /* if no more nodes, quit */

                temp = (*mylstPtr)->next ;  /* keep next node reference*/
                (*mylstPtr)-> next = NULL;  /* Since current node to be
                                        appended as last to either evenlist or oddlist */

                if ((*mylstPtr)->data % 2 == 0) { /*even node */
                        append (evenlstPtr, *mylstPtr);
                        /* note: expression &(*evenlstPtr) same as evelstPtr*/
                }
                else { /* odd node */
                        append (oddlstPtr, *mylstPtr);
                }
                *mylstPtr = temp;
        }
        return ;
}
```

Για περαιτέρω εξάσκηση

1. Να γράψετε πλήρη προγράμματα C που να χρησιμοποιούν τις ανωτέρω συναρτήσεις και να επιδεικνύουν τη χρήση τους.
2. Να γράψετε αναδρομικές υλοποιήσεις των brklst και insertsorted.
3. Να τροποποιήσετε τις συναρτήσεις brklst και insertsorted, ώστε να λαμβάνουν ως παράμετρο το κριτήριο του διαχωρισμού σε δύο λίστες ή το κριτήριο ταξινόμησης αντίστοιχα. Ζητείται να χρησιμοποιήσετε συναρτήσεις callback.