## Il compito del 11 novembre 2004

Alcuni errori frequenti riscontrati durante la correzione del compito:

1. In alcuni elaborati il typedef/struct non è stato utilizzato correttamente

2. Molti hanno confuso la dichiarazione di un array, l'accesso ai suoi componenti, ed il passaggio di tale array come parametro ad una funzione

3. Molti hanno confuso la differenza tra "chiamata per riferimento" e "chiamata per valore"…

4. Molti hanno confuso puntatori a determinati valori ed i valori stessi

## Il compito del 11 novembre 2004

Un commerciante vuole tenere sotto controllo i prezzi dei suoi prodotti. A tal scopo vuole realizzare un programma che chieda, per un numero finito di prodotti, il nome del prodotto e il suo prezzo in euro, e restituisca come output il nome del prodotto più costoso e il prezzo incriminato.

A tal scopo si definisca inizialmente una struttura di nome prodotto, composta dal nome del prodotto (al più 127 caratteri) e dal prezzo (rappresentato tramite float).

## Il compito del 11 novembre 2004

Si realizzi poi:
a) una procedura

```
void calcola( prodotto lista[], int length, char
          piucaro[], float *prezzo)
```

che riceve in ingresso un array `lista` di tipo `prodotto`, e la lunghezza `length` di tale array. La funzione deve determinare il prodotto più costoso e copiarne il nome nella stringa `piucaro`; il costo in euro deve poi essere restituito tramite la variabile `prezzo` (passata per riferimento). Al fine di copiare il nome del prodotto più costoso, si utilizzi la funzione di libreria `strcpy(char *s, char *ct)`, che copia la stringa `ct` nella stringa `s`, terminatore compreso. (punti 6)

## Il compito del 11 novembre 2004

b) un programma `main()` che chieda all'utente di inserire, alternativamente, il nome di un prodotto e il suo prezzo, per un totale di `DIM` prodotti (si controlli che il prezzo sia maggiore o uguale a 0; in caso contrario si ripeta la richiesta di inserire il prezzo). Il programma deve provvedere a memorizzare tali informazioni in un array di tipo `prodotto` precedentemente definito, di dimensione fissa `DIM` (si supponga per semplicità che `DIM` valga 10). Una volta raccolte le informazioni, si utilizzi la procedura `calcola()` definita al punto precedente per ottenere il nome e il prezzo del prodotto più costoso, e si stampino a video tali informazioni. (punti 6)

# Il compito del 11 novembre 2004

```
#include <stdio.h>
#include <string.h>

#define MAX 128
#define DIM 10

typedef struct {
    char nome[MAX];
    float prezzo;
} prodotto;
```

# Il compito del 11 novembre 2004

```
void calcola(prodotto lista[], int length, char piucaro[], float *
prezzo) {

     int i;
     int pos = 0;

     if (length <= 0) {  // controllo non richiesto dal testo...
          strcpy(piucaro, "ERRORE");
          *prezzo = 0;
          return;
     }
     for (i=1; i<length; i++)
          if (lista[i].prezzo > lista[pos].prezzo) {
               pos = i;
          }
     strcpy(piucaro, lista[pos].nome);
     *prezzo = lista[pos].prezzo;
}
```

# Il compito del 11 novembre 2004

```
int main() {
      int i;
      prodotto lista[DIM];
      char piucaro[MAX];
      float prezzo = 0;

      for (i=0; i<DIM; i++) {
            printf("Inserire nome prodotto: ");
            scanf("%s", lista[i].nome);
            do {
                  printf("Inserire prezzo: ");
                  scanf("%f", &lista[i].prezzo);
            } while (lista[i].prezzo < 0);
      }

      calcola(lista, i, piucaro, &prezzo);

      printf("Il prodotto piu' caro e' il %s, che costa %6.2f
euro.\n", piucaro, prezzo);

      return 0;
}
```

# Il compito del 11 novembre 2004

Esercizio 4 (punti 6)

Si scriva una funzione ricorsiva `int somma(int x)` che, ricevuto come parametro un numero intero positivo (pari o dispari), calcoli ricorsivamente la somma di tutti i numeri pari compresi tra 0 e quel numero (compreso).

## Il compito del 11 novembre 2004

```
int somma(int x) {

 if (x == 0)
   return 0;

 else {
    if (x%2 == 0)
      return x + somma(x  2);
    else
      return somma(x- 1);
 }
}
```

## Es1: ADT e allocazione dinamica della memoria

Si vuole realizzare un programma per eseguire calcoli elementari su numeri complessi. A tal scopo si definisca una opportuna struttura dati per rappresentare i numeri complessi, con relative operazioni di **somma** e **sottrazione**.

Si definiscano poi due funzioni, una per *stampare* ed una per *leggere* numeri complessi, che ricevano come parametro d'ingresso almeno un puntatore a FILE.

# Es1: ADT e allocazione dinamica della memoria

Si scriva poi un programma che legge da stdin un numero intero N; il programma provvederà poi a chiedere all'utente N numeri complessi che saranno salvati temporaneamente in un array.

Il programma chieda infine il nome di un file di testo all'utente, e provveda a scrivere su tale file i numeri complessi inseriti.

Si discuta infine eventuali varianti nel caso in cui invece di leggere i numeri complessi da stdin, si voglia utilizzare un file contenente un numero imprecisato di elementi.

# Es1: ADT e allocazione dinamica della memoria

## ADT complex_number

```
struct complex {
  float re;
  float im;
};

typedef struct complex complex_number;


complex_number create(float real, float imm) {
  complex_number result;
  result.re = real;
  result.im = imm;
  return result;
}

float getReal(const complex_number num) {
  return num.re;
}


float getIm(const complex_number num) {
  return num.im;
}
```

# Es1: ADT e allocazione dinamica della memoria

## ADT complex_number

```
complex_number sum( const complex_number a,
                    const complex_number b) {
  complex_number result;
  result = create(getReal(a)+getReal(b), getIm(a)+getIm(b));
  return result;
}

complex_number dif( const complex_number a,
                    const complex_number b) {
  complex_number result;
  result = create(getReal(a)  getReal(b), getIm(a)  getIm(b));
  return result;
}


void printComplexNumber(const complex_number a, FILE * f) {
  fprintf(f, "%f:%f\n", getReal(a), getIm(a));
}
```

# Es1: ADT e allocazione dinamica della memoria

## ADT complex_number

```
int scanComplexNumber(FILE * f, complex_number * result) {
  int val;
  float real;
  float imm;

  val = fscanf(f, "%f:%f", &real, &imm);
  if (val != EOF) {
    *result = create(real, imm);
    return 0;
  }
  else
    return 1;
}
```

# Es1: ADT e allocazione dinamica della memoria

## ADT complex_number

```
int main() {

  int dim;
  int i;
  int j;
  int result;
  complex_number * V;
  complex_number temp;
  char nomefile[MAX_DIM];
  FILE * f;


  printf("Inserire dimensione vettore: ");
  scanf("%d", &dim);

  V = (complex_number*) malloc(sizeof(complex_number) * dim);
  for (i=0; i<dim; i++) {
    printf("Inserire parte reale ed immaginaria, xxx:yyy : ");
    result = scanComplexNumber(stdin, &temp);
    if (result == 0)
          V[i] = temp;
    else
          V[i] = create(0,0);
  }
```

# Es1: ADT e allocazione dinamica della memoria

## ADT complex_number

```
  ...

  printf("Nome file su cui salvare: ");
  scanf("%s", nomefile);

  if ((f=fopen(nomefile, "w")) == NULL) {
    printf("Errore   durante   l'apertura   del   file   %s.",
  nomefile);
    exit(  1);
  }

  for (i=0; i<dim; i++)
    printComplexNumber(V[i], f);
  fclose(f);

  free(V);
  return 0;
}
```

# Es2: ADT
# Astrazione di vettore infinito

Si progetti un programma capace di leggere da stdin un numero (teoricamente) infinito di interi positivi, terminati da uno 0.

A tal scopo si realizzi un ADT che implementi l'idea di un vettore di interi a dimensione infinita, utilizzando apposite strutture e array allocati dinamicamente.

Devono essere inoltre definite funzioni e predicati per:

a) Creare un vettore vuoto

b) Aggiungere un intero in coda al vettore

c) Ottenere la dimensione (logica) del vettore

d) Ottenere un elemento del vettore data la sua posizione

# Es2: ADT
# Astrazione di vettore infinito

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

#define FIRST_SIZE 5
#define INCREMENT_SIZE 5


typedef struct {
  int p_dim;            // physical dimension
  int l_dim;            // logical dimension
  int * pointer;   // pointer to the data
} array_int;


array_int createEmpty();
int size(array_int a);
void add(int num, array_int * a);
int get(const array_int a, int pos);
```

# Es2: ADT
## Astrazione di vettore infinito

```
array_int createEmpty() {
  array_int result;
  result.pointer = malloc( sizeof(int) * FIRST_SIZE);
  result.p_dim = FIRST_SIZE;
  result.l_dim = 0;
  return result;
}


int size(array_int a) {
  return a.l_dim;
}
```

# Es2: ADT
## Astrazione di vettore infinito

```
void add(int num, array_int * a) {
  int * temp;
  int i;

  if (size(*a) < (a->p_dim)) {
    (a->pointer)[size(*a)] = num;
    (a->l_dim) = (a->l_dim) + 1;
  }
  else {
    temp = a->pointer;
    (a->pointer)
         = malloc(sizeof(int) * ((a->p_dim) +
  INCREMENT_SIZE));

    for (i=0; i<(a->l_dim); i++)
         (a->pointer)[i] = temp[i];

    (a->pointer)[a->l_dim] = num;
    (a->l_dim) = (a->l_dim) + 1;
    free(temp);
  }
}
```

## Es2: ADT
## Astrazione di vettore infinito

```
int get(const array_int a, int pos) {
 if (pos < 0)
   pos = 0;

 if (size(a) <= pos)
   pos = size(a)-  1;

 if (size(a) == 0)
   return- 1;

 return (a.pointer)[pos];
}
```

## Es2: ADT
## Astrazione di vettore infinito

```
int main() {

 int i;
 int num;
 array_int V = createEmpty();

 printf("Inserisci i numeri, 0 per terminare:
 ");
 scanf("%d", &num);
 while (num >0) {
   add( num, &V);
   scanf("%d", &num);
 }
 for (i=0; i<size(V); i++) {
   printf("%d\n", get(V, i));
 }
}
```