REPORT
TRACCIA GIORNO 3

Obiettivo:

L'obiettivo dell'esercizio datoci è:

- Descrivere il funzionamento del programma prima dell'esecuzione;
- Riprodurre ed eseguire il programma;
- Modificare affinché si verifichi un errore di segmentazione.

Il codice che ci è stato dato è il seguente:

```
#include <stdio.h>

int main () {

int vector [10], i, j, k;
int swap_var;


printf ("Inserire 10 interi:\n");

for ( i = 0 ; i < 10 ; i++)
        {
        int c= i+1;
        printf("[%d]:", c);
        scanf ("%d", &vector[i]);
        }


printf ("Il vettore inserito e':\n");
for ( i = 0 ; i < 10 ; i++)
        {
        int t= i+1;
        printf("[%d]: %d", t, vector[i]);
        printf("\n");
        }


for (j = 0 ; j < 10 - 1; j++)
        {
        for (k = 0 ; k < 10 - j - 1; k++)
                {
                        if (vector[k] > vector[k+1])
                        {
                        swap_var=vector[k];
                        vector[k]=vector[k+1];
                        vector[k+1]=swap_var;
                        }
                }
        }
printf("Il vettore ordinato e':\n");
for (j = 0; j < 10; j++)
        {
        int g = j+1;
        printf("[%d]:", g);
        printf("%d\n", vector[j]);
        }

return 0;


}
```

ANALISI DEL CODICE

Dopo un'attenta analisi del codice, abbiamo detotto che fosse un codice in linguaggio C, il quale, chiede all'utente di inserire 10 numeri interi che verranno poi memorizzati dentro un array chiamato **vector**. La prima stampa dell'array, mostra tutti i numeri per come sono stati inseriti dall'utente, mentre la seconda, riordina i numeri. Infine viene stampato il vettore in ordine crescente.

Una volta eseguito il codice, possiamo dire che le ipotesi sul suo funzionamento fossero corrette, unica cosa da aggiungere per quanto riguarda il riordinamento dei numeri del vettore in ordine crescente, che ciò viene fatto grazie all'algoritmo **Bubble Sort**. Un algoritmo che scambia e confronta ripetutamente elementi di un array fino a che l'intera sequenza non è ordinata.

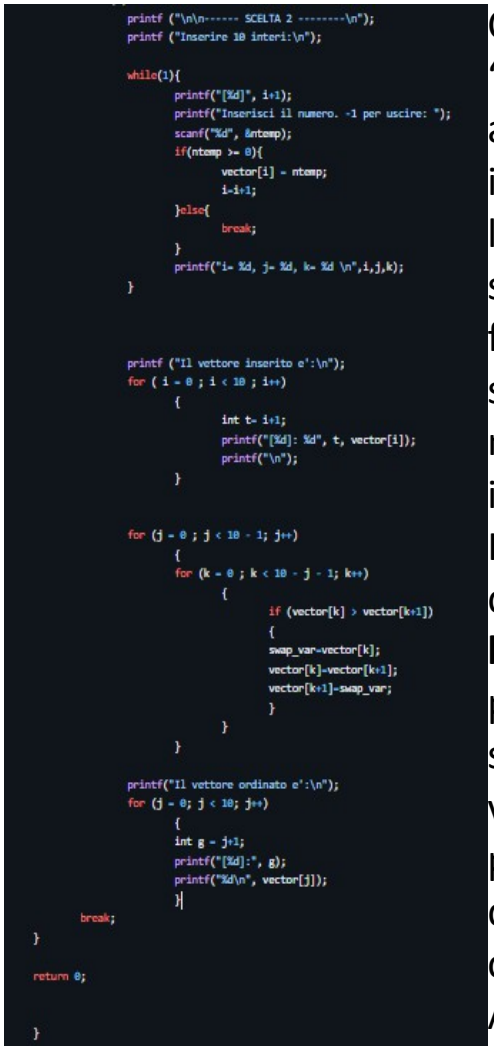
MODIFICA DEL CODICE CON ERRORE DI SEGMENTAZIONE

Il terzo punto della traccia chiedeva di modificare il codice affinché avvenga un'errore di segmentazione, ovvero un'errore che si crea quando il programma, nel nostro caso il vettore, prova ad accedere ad un'area di memoria non consentita, molte volte l'errore di segmentazione si verifica anche quando si tenta di accedere alla memoria con metodi non consentiti.

```c
#include <stdio.h>

int main () {

int vector[10], i, j, k, ntemp, scelta;
int swap_var;
i=0;
j=0;
k=0;

while(1){
        printf("------- MENU --------\n");
        printf("1) Programma corretto\n");
        printf("2) Programma con possibile BOF\n");
        printf("---------------------\n");
        printf("Scelta: ");
        scanf("%d", &scelta);
        if(scelta > 2){
                printf("Inserisci un valore ammesso.\n\n");
        }else{
                break;
        }
}
```

Per la corretta esecuzione dell'esercizio, abbiamo deciso di creare un **Menù**, il quale permetterà all'utente di scegliere tra il **case 1**, che ci eseguirà il codice corretto senza errori di segmentazioni, che è lo stesso codice che ci è stato dato nella traccia dell'esercizio. Quindi andremo direttamente al **case 2**, dato che abbiamo anche cambiato qualcosa nel codice.

CASE 2



```
printf ("\n\n------ SCELTA 2 --------\n");
printf ("Inserire 10 interi:\n");

while(1){
        printf("[%d]", i+1);
        printf("Inserisci il numero. -1 per uscire: ");
        scanf("%d", &ntemp);
        if(ntemp >= 0){
                vector[i] = ntemp;
                i=i+1;
        }else{
                break;
        }
        printf("i= %d, j= %d, k= %d \n",i,j,k);
}



printf ("Il vettore inserito e':\n");
for ( i = 0 ; i < 10 ; i++)
        {
                int t= i+1;
                printf("[%d]: %d", t, vector[i]);
                printf("\n");
        }


for (j = 0 ; j < 10 - 1; j++)
        {
        for (k = 0 ; k < 10 - j - 1; k++)
                {
                        if (vector[k] > vector[k+1])
                        {
                        swap_var=vector[k];
                        vector[k]=vector[k+1];
                        vector[k+1]=swap_var;
                        }
                }
        }

printf("Il vettore ordinato e':\n");
for (j = 0; j < 10; j++)
        {
        int g = j+1;
        printf("[%d]:", g);
        printf("%d\n", vector[j]);
        }
        break;
}

return 0;


}
```

Come vediamo nell'immagine accanto, per “creare” l'errore di segmentazione, abbiamo aggiunto un ciclo while, il quale crea un ciclo in cui, l'utente inserisce i 10 numeri e, in caso l'utente volesse inserire meno di 10 numeri, se digitasse -1, il programma verebbe fermato, uscendo dal ciclo e visualizzando soltanto i numeri inseriti dall'utente, mentre nel primo codice l'utente era obbligato ad inserire 10 numeri.
In questo caso, l'utente può anche inserire più di 10 numeri in vector, così da creare un **buffer overflow**. Ovvero, se l'utente inserisse più di 10 numeri, il ciclo non verifica se **i** supera il limite dell'array, così facendo questi vengono scritti fuori dai limiti dell'array, potenzialmente sovrascrivendo altre aree della memoria, causando crash e comportamenti imprevedibili
Abbiamo inserito la variabile ntemp, una variabile che, legge il valore prima di inviarlo all'array, mente nel primo caso veniva mandato direttamente senza che venisse letto dal codice.
Inoltre, l'incremento di i avviene solo se il numero è >0, quindi se l'utente inserisse un numero negativo, il ciclo si interromperebbere.

CONCLUSIONI

In conclusione siamo riusciti a raggiungere tutti gli obiettivi richiesti dall'esercizio, riuscendo a capire cosa facesse il codice ancora prima di eseguirlo.
Siamo riusciti a modificare il codice mettendo di fronte all'utente la possibilità di scegliere tra un programma corretto e uno che presenta un'errore di segmentazione.