Analisi Programma in C - BOF

1. Descrizione del Funzionamento (Analisi Preliminare)

1.1 Scopo del Programma

Il programma implementa l'algoritmo di ordinamento **Bubble Sort** per ordinare un array di 10 numeri interi inseriti dall'utente. L'algoritmo confronta elementi adiacenti e li scambia se sono in ordine scorretto, ripetendo il processo fino a quando l'array non risulta ordinato.

1.2 Struttura e Logica

Fase 1: Input (Righe 6-11)

```
for ( i = 0 ; i < 10 ; i++)

      {

      int c= i+1;

      printf("[%d]:", c);

      scanf ("%d", &vector[i]);

      }
```

- Chiede all'utente di inserire 10 interi
- Utilizza un contatore `c = i+1` per visualizzare posizioni 1-10 (anziché 0-9)
- Memorizza i valori nell'array `vector[]`

Fase 2: Visualizzazione Input (Righe 14-20)

```
printf ("Il vettore inserito e':\n");

for ( i = 0 ; i < 10 ; i++)

        {

        int t= i+1;

        printf("[%d]: %d", t, vector[i]);

      printf("\n");

      }
```

- Stampa l'array così come inserito

Fase 3: Bubble Sort (Righe 23-35)

```
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;
                  }
            }
      }
```

- **Loop esterno (j)**: esegue 9 iterazioni (da 0 a 8)
- **Loop interno (k)**: confronta elementi adiacenti
- **Condizione**: se `vector[k] > vector[k+1]`, scambia i valori usando `swap_var`
- Ogni passata sposta l'elemento più grande alla fine

Fase 4: Output (Righe 36-42)

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

- Stampa l'array ordinato in ordine crescente

1.3 Ipotesi sul Funzionamento Atteso

Aspetto	Ipotesi Preliminare
Input	Accetterà 10 numeri interi senza restrizioni
Ordinamento	Produrrà array in ordine crescente (ascendente)
Memoria	Utilizzerà solo lo stack (array locale)
Termination	Completerà sempre in tempo finito

---

2. Esecuzione del Programma e Verifica delle Ipotesi

2.1 Test Case: Input Casuale

Input fornito:
5, 8, 9, 2 ,6 , 12, 10, 1, 3, 4

Esecuzione attesa:

```
Inserire 10 interi:
[1]:5
[2]:8
[3]:9
[4]:2
[5]:6
[6]:12
[7]:10
[8]:1
[9]:3
[10]:4
Il vettore inserito e':
[1]: 5
[2]: 8
[3]: 9
[4]: 2
[5]: 6
[6]: 12
[7]: 10
[8]: 1
[9]: 3
[10]: 4
Il vettore ordinato e':
[1]:1
[2]:2
[3]:3
[4]:4
[5]:5
[6]:6
[7]:8
[8]:9
[9]:10
[10]:12
```

2.2 Verifica delle Ipotesi

✅ **Ipotesi 1 - Input**: VERIFICATA

- Il programma accetta correttamente 10 interi

✅ **Ipotesi 2 - Ordinamento**: VERIFICATA

- L'array è ordinato in ordine crescente corretto
- L'algoritmo bubble sort funziona come previsto

✅ **Ipotesi 3 - Memoria**: VERIFICATA

- L'array è dichiarato come variabile locale
- Nessun utilizzo di allocazione dinamica

2.3 Osservazioni Importanti

1. **Robustezza limitata**: Il programma non valida l'input
2. **Buffer overflow**: Non controlla il numero di elementi
3. **Memoria stack**: Array fisso da 10 elementi

---

3. Modifica per Causare Segmentation Fault

3.1 Cause Comuni di Segmentation Fault

Un **Segmentation Fault** si verifica quando un programma tenta di accedere a memoria non autorizzata. Le cause tipiche sono:

- Dereferenziazione di puntatori NULL
- Accesso fuori dai limiti di un array (buffer overflow)
- Stack overflow per ricorsione infinita
- Scrittura in memoria protetta (read-only)

3.2 Modifica: Manipolazione Puntatori

Codice modificato che causa segmentation fault:

```c
#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]);
    }
```

Per provocare il segmentation fault abbiamo rimosso l'operatore & nella scanf alla riga 15, facendo sì che la funzione scriva in un indirizzo di memoria non valido invece che nell'elemento dell'array. Questo accesso in scrittura a memoria non autorizzata provoca il segnale di errore di segmentazione (segmentation fault) durante l'esecuzione del programma:

```
Inserire 10 interi:
[1]:2
zsh: segmentation fault  ./BOF
```

4. Codice ultimato

```c
#include <stdio.h>

#define N 10

/* ================ UTILITIES ================ */

/*
 * Stampa il contenuto di un vettore di interi
 * accompagnato da un messaggio descrittivo.
 */
void stampa_vettore(int v[], int n, const char *msg)
{
    printf("\n%s\n", msg);
    for (int i = 0; i < n; i++)
        printf("[%d]: %d\n", i + 1, v[i]);
}

/*
 * Ordina il vettore usando l'algoritmo Bubble Sort
 */
void bubble_sort(int v[], int n)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (v[j] > v[j + 1])
            {
                int tmp = v[j];
                v[j] = v[j + 1];
                v[j + 1] = tmp;
            }
}

/* ================ INPUT SICURO ================ */

/*
 * Versione corretta: input validato.
 * Accetta solo numeri interi e ripete la richiesta
 * finché l'input non è valido.
 */
void input_sicuro(int v[], int n)
{
    char buffer[100];
```

```c
    int value;
    char extra;

    printf("\n✅ Versione corretta (input sicuro)\n");
    printf("Inserisci %d numeri interi:\n", n);

    for (int i = 0; i < n; i++)
    {
        while (1)
        {
            printf("[%d]: ", i + 1);
            fgets(buffer, sizeof(buffer), stdin);

            if (sscanf(buffer, "%d %c", &value, &extra) == 1)
            {
                v[i] = value;
                break;
            }

            printf("❌ Input non valido. Inserisci SOLO un intero.\n");
        }
    }
}

/* ========== VULNERABILE DETERMINISTICA ========== */

/*
 * Versione vulnerabile con comportamento deterministico.
 * Dopo l'inserimento dell'11° valore forza una
 * dereferenziazione di puntatore NULL → segmentation fault.
 */
void input_vulnerabile_deterministica(int v[], int n)
{
    char buffer[100];
    int value;
    char extra;

    printf("\n⚠️ Versione vulnerabile DETERMINISTICA\n");
    printf("Inserisci %d numeri interi (crash all'11°):\n", n + 1);

    for (int i = 0; i <= n; i++)
```

```c
    {
        while (1)
        {
            printf("[%d]: ", i + 1);
            fgets(buffer, sizeof(buffer), stdin);

            if (sscanf(buffer, "%d %c", &value, &extra) == 1)
                break;

            printf("❌ Input non valido. Inserisci SOLO un intero.\n");
        }

        if (i < n)
        {
            v[i] = value;
        }
        else
        {
            int *p = NULL;
            *p = value;   // 💥 segmentation fault garantita
        }
    }
}

/* ======== VULNERABILE NON DETERMINISTICA ======== */

/*
 * Versione vulnerabile NON deterministica.
 * Scrive oltre i limiti dell'array causando
 * Undefined Behavior: il programma può
 * funzionare, corrompersi o crashare.
 */
void input_vulnerabile_nondeterministica(int v[], int n)
{
    char buffer[100];
    int value;
    char extra;

    printf("\n⚠️ Versione vulnerabile NON deterministica\n");
    printf("Inserisci 10 numeri interi:\n");

```

```c
    for (int i = 0; i < n + 10; i++)   // scrittura fuori dai limiti
    {
        while (1)
        {
            printf("[%d]: ", i + 1);
            fgets(buffer, sizeof(buffer), stdin);

            if (sscanf(buffer, "%d %c", &value, &extra) == 1)
                break;

            printf("❌ Input non valido. Inserisci SOLO un intero.\n");
        }

        v[i] = value;   // Undefined Behavior
    }
}

/* ================== MAIN ================== */

/*
 * Funzione principale:
 * - permette la scelta della modalità
 * - gestisce il flusso del programma
 * - stampa e ordina il vettore
 */
int main()
{
    int vector[N];
    int scelta;

    printf("Scegli la modalità:\n");
    printf("1️⃣  Versione corretta\n");
    printf("2️⃣  Versione vulnerabile deterministica\n");
    printf("3️⃣  Versione vulnerabile NON deterministica\n");
    printf("> ");

    scanf("%d", &scelta);
    while (getchar() != '\n'); // pulizia buffer

    switch (scelta)
    {
```

```c
    {
        case 1:
            input_sicuro(vector, N);
            break;
        case 2:
            input_vulnerabile_deterministica(vector, N);
            break;
        case 3:
            input_vulnerabile_nondeterministica(vector, N);
            break;
        default:
            printf("Scelta non valida.\n");
            return 1;
    }

    stampa_vettore(vector, N, "Vettore inserito:");
    bubble_sort(vector, N);
    stampa_vettore(vector, N, "Vettore ordinato:");

    return 0;
}
```

Abbiamo ultimato il nostro codice inserendo un menu' che permette di scegliere all'utente 3 versioni del programma (in tutte le versioni vengono rifiutati input diversi da numeri interi):

1 - Versione Corretta

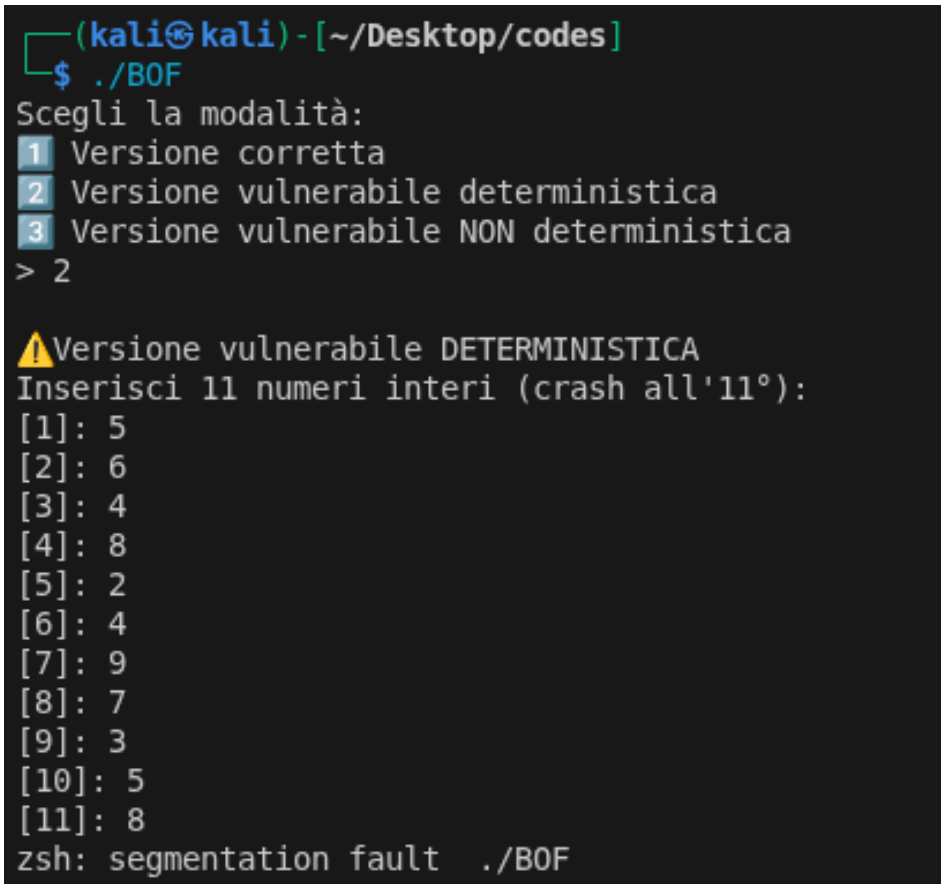
Versione standard del programma che esegue l'algoritmo di ordinamento **Bubble Sort** per ordinare un array di 10 numeri

```
┌──(kali㉿kali)-[~/Desktop/codes]
└─$ ./BOF
Scegli la modalità:
1️⃣ Versione corretta
2️⃣ Versione vulnerabile deterministica
3️⃣ Versione vulnerabile NON deterministica
> 1

✅ Versione corretta (input sicuro)
Inserisci 10 numeri interi:
[1]: 3
[2]: 8
[3]: 5.6
❌ Input non valido. Inserisci SOLO un intero.
[3]: code
❌ Input non valido. Inserisci SOLO un intero.
[3]: 8
[4]: 9
[5]: 7
[6]: 2
```

2 - Versione Vulnerabile Deterministica

Versione che causa segmentation fault all'11° input dell'utente



3 - Versione Vulnerabile NON Deterministica

Versione che causa BOF (Buffer OverFlow) al programma scrivendo piu' dati in un'area di memoria temporanea (Buffer) causando il crash

```
┌──(kali㉿kali)-[~/Desktop/codes]
└─$ ./BOF
[1]: 5
[2]: 6
[3]: 9
[4]: 8
[5]: 44
[6]: 1
[7]: 2
[8]: 3
[9]: 5
[10]: 6
[11]: 7
[12]: 1
[13]: 2
[14]: 3
[15]: 6
[16]: 5
[17]: 8
[18]: 4
[19]: 25
[20]: 54

Vettore inserito:
[1]: 5
[2]: 6
[3]: 9
[4]: 8
[5]: 44
[6]: 1
[7]: 2
[8]: 3
[9]: 5
[10]: 6

Vettore ordinato:
[1]: 1
[2]: 2
[3]: 3
[4]: 5
[5]: 5
[6]: 6
[7]: 6
[8]: 8
[9]: 9
[10]: 44
zsh: segmentation fault  ./BOF
```

4. Conclusioni

4.1 Risultati della Verifica Preliminare

- L'algoritmo bubble sort funziona correttamente per input validi (10 interi)
- L'ordinamento è accurato e nell'ordine crescente atteso
- Il programma non ha protezioni contro input errati

4.2 Vulnerabilità Identificate

Buffer Overflow: Nessun controllo sulla dimensione dell'input
Validazione Input: Nessun controllo sui valori letti
Fixed Size: Array hardcoded a 10 elementi

4.3 Lezioni Apprese

1. **Buffer overflow** è una delle cause più comuni di segmentation fault
2. Programmazione sicura richiede **validazione degli input**
3. Array statici rappresentano un **rischio di sicurezza**
4. Uso di **costanti simboliche** è una best practice