Esercitazione 08

2025/10/27 – Funzioni

L. Alessandrini

Credits: A. Montenegro per gli esercizi

Esercizio 08

Rettangolo: Appartenenza, Perimetro e Area

Testo

Si definisca un tipo di dato che rappresenti un punto nel piano cartesiano. Si definisca un altro tipo di dato che rappresenti quattro vertici di un rettangolo nel piano cartesiano.

Si scriva un programma che:

1. chiede in input un punto
2. chiede in input un rettangolo
3. verifica se il punto si trova all'interno del rettangolo
4. calcola area e perimetro del rettangolo

Si scrivano le funzioni:

- `leggiPunto()`: che legge le coordinate di un punto e restituisce tale punto
- `leggiRettangolo()`: che legge i punti di un rettangolo e restituisce un rettangolo. Imporre dei controlli per verificare che i punti inseriti siano effettivamente parte di un rettangolo.
- `nelRettangolo()`: che dato un punto e un rettangolo, restituisce 1 se il punto è all'interno del rettangolo e 0 altrimenti
- `calcolaArea()`: che dato un rettangolo restituisce la sua area
- `calcolaPerimetro()`: che dato un rettangolo restituisce il suo perimetro
- `calcolaAeraPerimetro()`: che dato un rettangolo, non restituisce nulla, ma fa in modo che la funzione chiamante abbia visione del perimetro e dell'area del rettangolo

Funzioni

Esercizi – Parte 2

Esercizio 09

Conta Occorrenze in un Array

Testo

Scrivere un programma che legge un array di interi e un numero e stampa il numero di volte che il numero appare all'interno dell'array.

Scrivere le funzioni:

- `leggiInt()`: che legge e restituisce un intero
- `leggiIntN()`: che legge e restituisce un intero minore di N e positivo
- `leggiArrayInt()`: che legge e restituisce un array di interi di una certa dimensione
- `contaOccorrenze()`: che prende in input un array e un numero e restituisce il numero di volte che il numero appare all'interno dell'array

Esercizio 10

Elimina Carattere da Stringa

Testo

Scrivere un programma che legge una stringa senza spazi e un carattere e stampa la stringa inserita senza il carattere inserito.

Scrivere le funzioni (da mappare con quanto possibile fare in C):

- `leggiCarattere()`: che legge e restituisce un carattere
- `leggiStringa()`: che legge e restituisce una stringa
- `rimuoviCarattere()`: che prende in input una stringa e un carattere e restituisce la stringa senza il carattere inserito (in un'altra stringa)

Esercizio 11

Lunghezza Massima Sottosequenza Ascendente

Testo

Scrivere un programma che prende in input nella funzione main una stringa. Il programma stampa la lunghezza della più lunga sottosequenza di caratteri ordinati in ordine ascendente nella stringa.

Scrivere le funzioni:

- `lunghezzaMassimaSottosequenzaAscendente()`: che prende in input un puntatore che punta alla stessa area di memoria della stringa e la lunghezza della stringa. La funzione restituisce un intero pari alla più lunga sottosequenza di caratteri ordinati in ordine ascendente nella stringa.
- `leggiStringa()`: che legge e restituisce una stringa

Esercizio 12

Elevare al Quadrato una Porzione dell'Array

Testo

Scrivere un programma che legge un array di lunghezza definita dall'utente (al massimo una dimensione predeterminata) e che modifica l'array elevando al quadrato una porzione di esso a partire da una posizione decisa dall'utente fino a quando l'array non termina o fino a quando non si incontra uno 0.

Scrivere le funzioni:

- `leggiInt()`: che legge e restituisce un intero
- `leggiIntN()`: che legge e restituisce un intero in [1,N]
- `leggiArrayInt()`: che legge e restituisce un array di interi di una certa dimensione
- `foo()`: che eleva al quadrato gli elementi dell'array da una certa posizione in poi fino ad arrivare alla fine dell'array o fino a quando l'array non termina. Tale funzione fa uso della funzione ausiliaria `subpow()`
- **`subpow()`: che prende l'inizio di una porzione dell'array e la dimensione di tale sotto-array e lo eleva al quadrato**
- `stampaArray()`: che stampa un array passato in input

Esercizio 12

Elevare al Quadrato una Porzione dell'Array

Testo

Scrivere un programma che legge un array di lunghezza definita dall'utente (al massimo una dimensione predeterminata) e che modifica l'array elevando al quadrato una porzione di esso a partire da una posizione decisa dall'utente fino a quando l'array non termina o fino a quando non si incontra uno 0.

Scrivere le funzioni:

- `leggiInt()`: che legge e restituisce un intero
- `leggiIntN()`: che legge e restituisce un intero in [1,N]
- `leggiArrayInt()`: che legge e restituisce un array di interi di una certa dimensione
- `foo()`: che eleva al quadrato gli elementi dell'array da una certa posizione in poi fino ad arrivare alla fine dell'array o fino a quando l'array non termina. Tale funzione fa uso della funzione ausiliaria `subpow()`
- **`subpow()`: che prende l'inizio di una porzione dell'array e la dimensione di tale sotto-array e lo eleva al quadrato NB: tutto quel pezzo dell'array! Partiamo da qui!**
- `stampaArray()`: che stampa un array passato in input

Esercizio 13

Elimina Sottostringa da Stringa

Testo

Scrivere un programma che prende in input una stringa e una seconda stringa. Il programma rimuove dalla prima stringa tutte le sequenze di caratteri uguali alla seconda stringa.

Scrivere le funzioni:

- `leggiStringa()`
- `rimuoviStringa()`

Esercizio 14

Shift di Matrice per Righe

Testo

Scrivere un programma che legge una matrice 3x4 ed effettua uno shift per righe verso il basso e stampa il risultato. L’ultima riga prende la posizione della prima riga.

Scrivere le funzioni:

- `leggiArray()`
- `leggiMatrice()`
- `stampaArray()`
- `stampaMatrice()`
- `copiaArray()`
- `shiftMatrice()`

Esercizio 15

Implementazione strcpy

Testo

Si implementi la funzione `strcpy(dest, src)`.

Puntatori

Es. Pag. 58 di teoria

Array e Puntatori

Modifica dei puntati

Si consideri il seguente programma:

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

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p + i) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

Array e Puntatori

Modifica dei puntati

Ora ne estraiamo i punti fondamentali per analizzarlo meglio

Array e Puntatori

Modifica dei puntati

Si consideri il seguente programma:

Copio nel puntatore l'indirizzo di `vet[0]`.

Coincide con

`p = &vet[0]`

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}

// [print valori di vet + return]
```

Array e Puntatori

Modifica dei puntati

Si consideri il seguente p

Vado a scrivere nella cella vet[i]:

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}

// [print valori di vet + return]
```

Array e Puntatori

Modifica dei puntati

Si consideri il seguente
p

Vado a scrivere nella cella vet[i]:

- p è come vet (`vet[0]`, dove `0` è l'offset partendo dall'indirizzo `vet`),

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}

// [print valori di vet + return]
```

Array e Puntatori

Modifica dei puntati

Si consideri il seguente p

Vado a scrivere nella cella vet[i]:

- p è come vet (`vet[0]`, dove `0` è l'offset partendo dall'indirizzo `vet`),
- i è l'offset partendo da p (`vet`)

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}

// [print valori di vet + return]
```

Array e Puntatori

Modifica dei puntati

Si consideri il seguente programma:

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}

// [print valori di vet + return]
```

Nota: non sto mai cambiando p!

Array e Puntatori

Modifica dei puntati

Si consideri il seguente programma:

```
[...]
int vet[L] = {-1,-1,-1,-1,-1};
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}

// [print valori di vet + return]
```

Arr

Modi

Si cor
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffe1d0)
valori di vet: [0 2 4 6 8 ]
```

```
for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]
}
```

Arr

Modi

Si cor
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffe1d0)
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]
}
```

Arr

Modi

Si cor
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffe1d0)
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
[print i, vet[i], *p,
p] (for)
```

```
for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]
}
```

Arr

Modi

Si cor
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffe1d0)
valori di vet: [0 2 4 6 8 ]
```

Come vedete, la linea evidenziata non modifica affatto p

```
[pr
```

```
[print i, vet[i], *p,
p] (for)
```

```
for(i = 0; i < L; i++){
   *(p + i) = 2*i;
   // [print i, vet[i], *p, p]
}
```

Arr

Modi

Si cor
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffe1d0)
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
[
p
```

Accede invece a p+i, usando il contatore del loop (i) come offset dall'indirizzo p (p=vet=vet[0])

```
for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]
}
```

Arr

Modi

Si cor
progr

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=0, (p=0x7fffffffe1d0)
i = 1, vet[1] = 2, *p=0, (p=0x7fffffffe1d0)
i = 2, vet[2] = 4, *p=0, (p=0x7fffffffe1d0)
i = 3, vet[3] = 6, *p=0, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=0, (p=0x7fffffffe1d0)
valori di vet: [0 2 4 6 8 ]
```

```
[print valori di vet]
```

```
[print i, vet[i], *p, p] (for)
```

```
[print valori di vet + return]
```

```
for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]
}
```

Array e Puntatori

Modifica dei puntati

Modifichiamo il programma … Cambiamo dove andiamo a scrivere all’interno del loop for

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

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p + i) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

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

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p++) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nindirizzo di i: %p", &i);
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

Ovvero cambio da:

```c
*(p + i) = 2*i;
```

a:

```c
*(p++) = 2*i;
```

Ed aggiungo la print dell'indirizzo di `i`

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

1. Scrivo nella cella puntata da p il valore 2*i
2. Incremento p

- La prima volta, scriverò in vet[0]=0, poi incremento p
- La seconda, scrivo in vet[1]=2
- […]

E l’ultima? Dopo aver incrementato p, a cosa punterà?

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
```

Notiamo innanzitutto come gli indirizzi aumentino di 4 parole

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7ffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7ffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7ffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7ffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7ffffffe1d4)
indirizzo di i: 0x7ffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

p++ incrementa dopo!
Prima viene eseguito vet[4]=8, poi a p viene aggiunto sizeof(int), e...

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

Dopo l'assegnazione nell'ultima iterazione punta allo stesso indirizzo di `i`, per questo *p=4 (=i)!

`p++` incrementa dopo! Prima viene eseguito `vet[4]=8`, poi a p viene aggiunto `sizeof(int)`, e…

```
v                                -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]=-1
fe1c4   vet[1]=-1
fe1c8   vet[2]=-1
fe1cc   vet[3]=-1
fe1d0   vet[4]=-1
fe1d4       i


            p
```

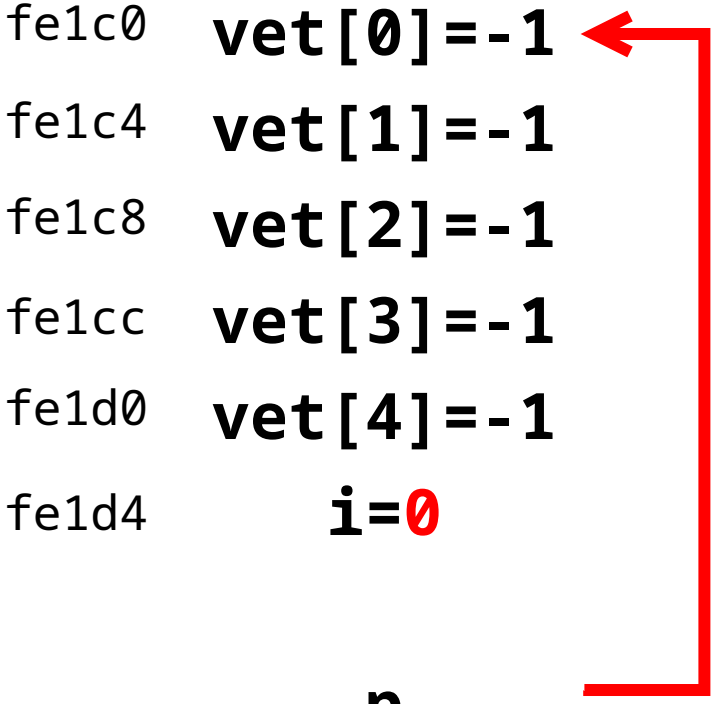
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

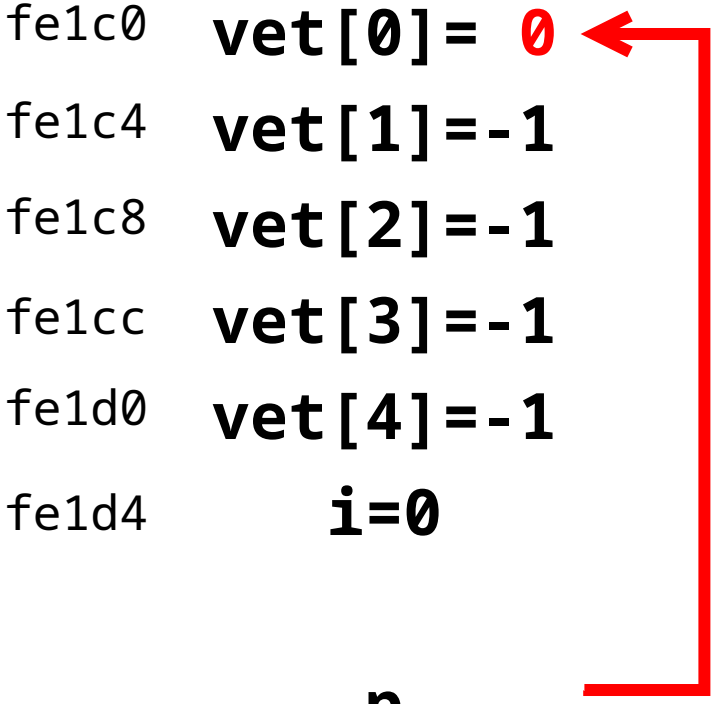
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

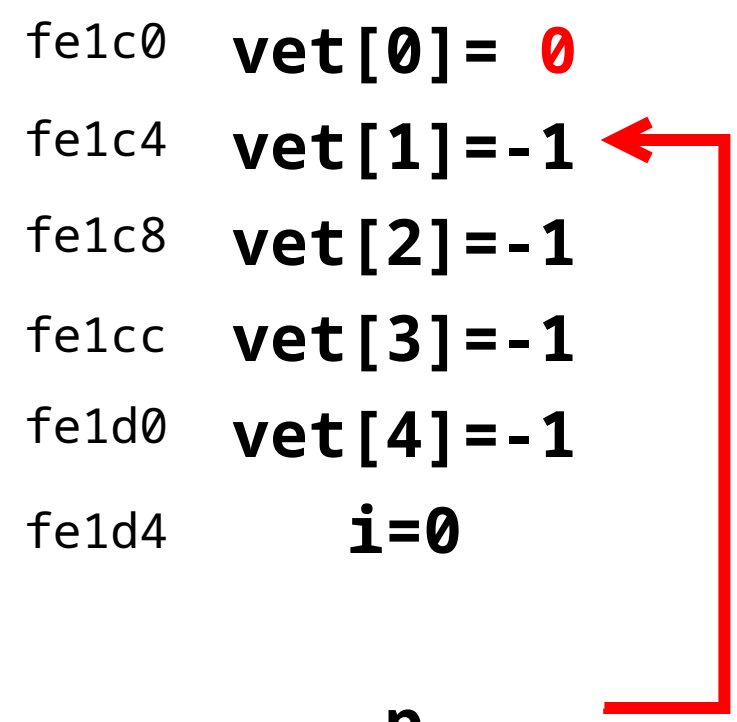
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

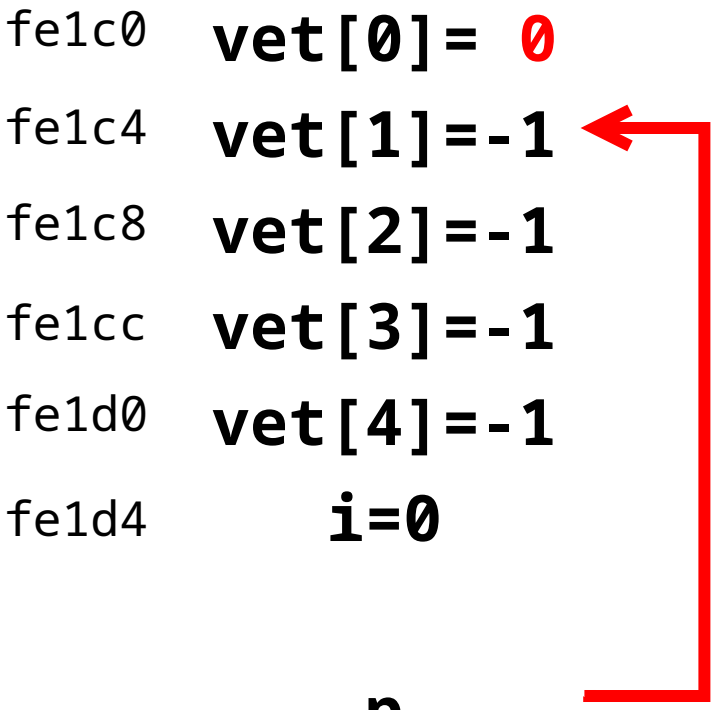
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

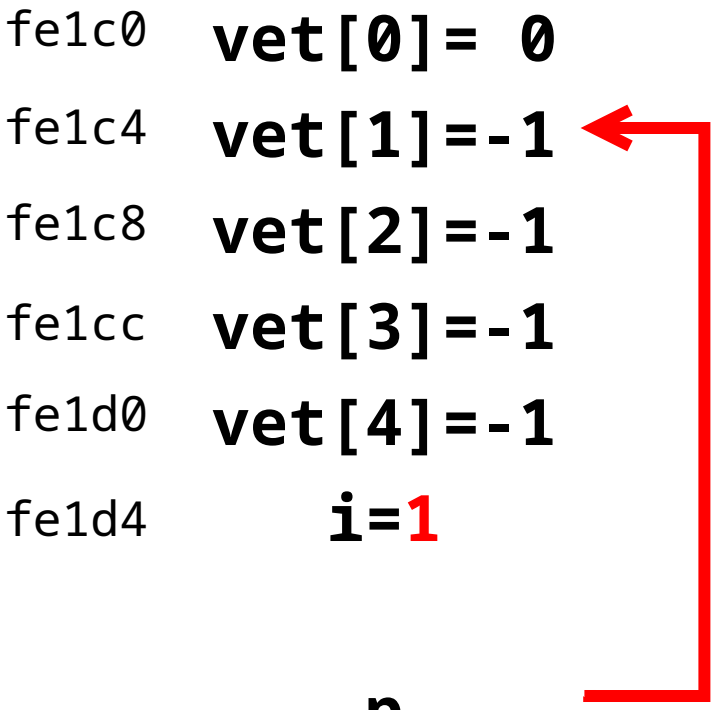
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

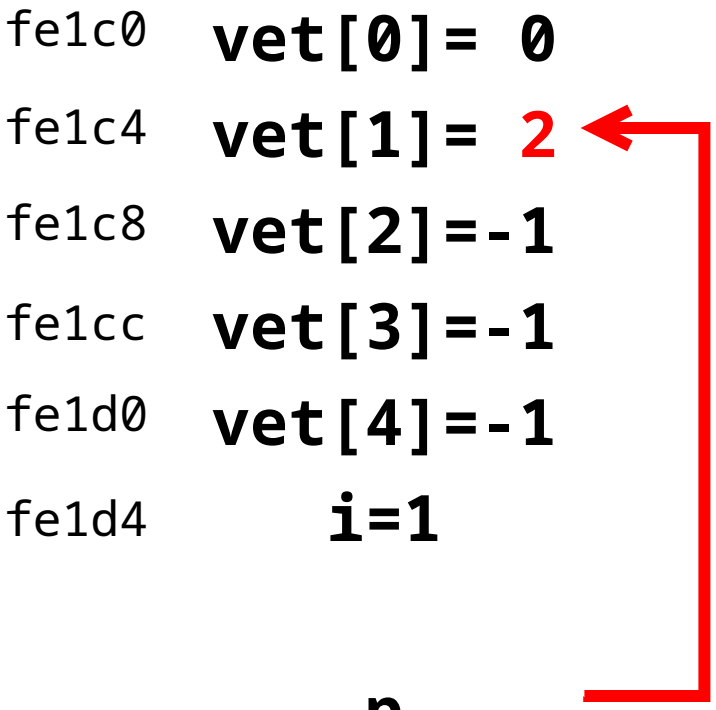
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

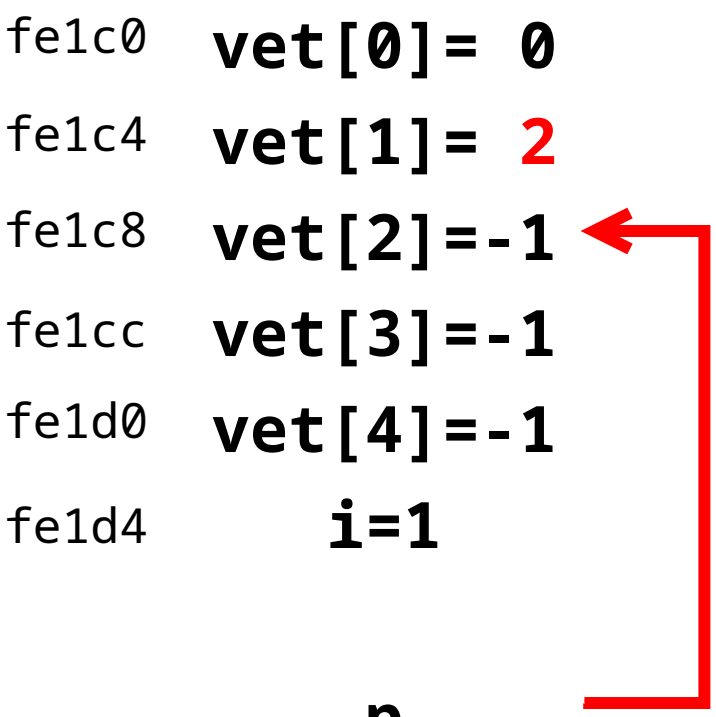
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]=-1
fe1cc   vet[3]=-1
fe1d0   vet[4]=-1
fe1d4      i=1


           p
```

i = 1

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]=-1
fe1cc   vet[3]=-1
fe1d0   vet[4]=-1
fe1d4      i=2

           p
```

i = 2

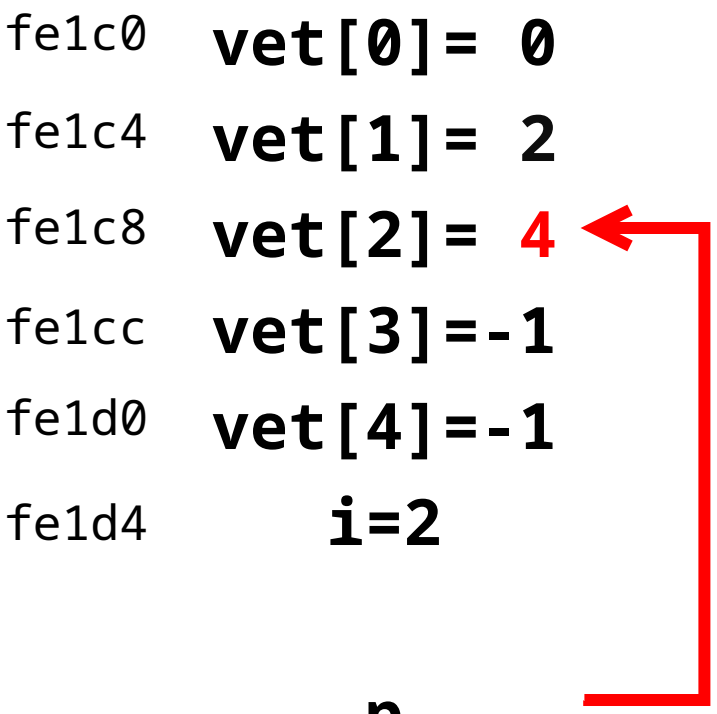
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 2

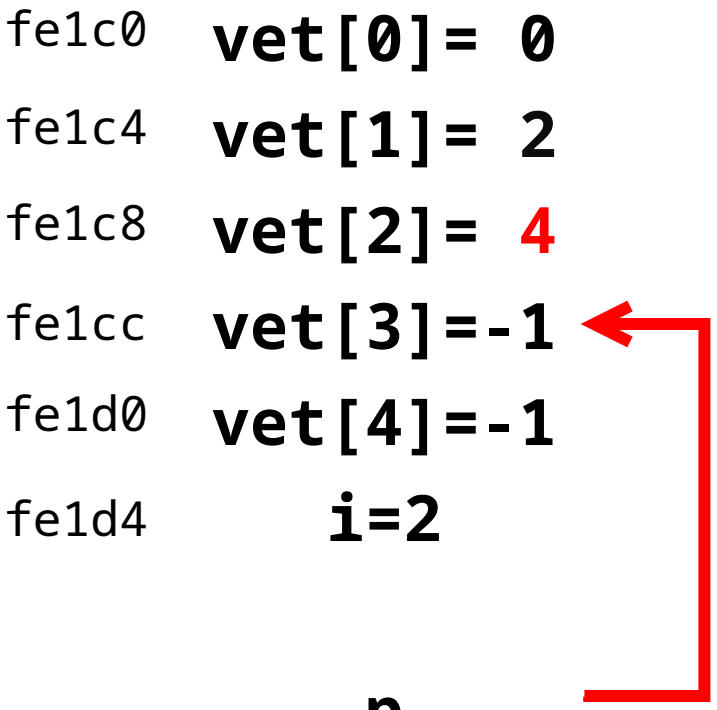
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 2

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7ffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7ffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7ffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7ffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7ffffffe1d4)
indirizzo di i: 0x7ffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]=-1
fe1d0   vet[4]=-1
fe1d4      i=2


           p
```

i = 2

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]=-1
fe1d0   vet[4]=-1
fe1d4      i=3


           p
```

i = 3

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]= 6
fe1d0   vet[4]=-1
fe1d4      i=3


           p
```

i = 3

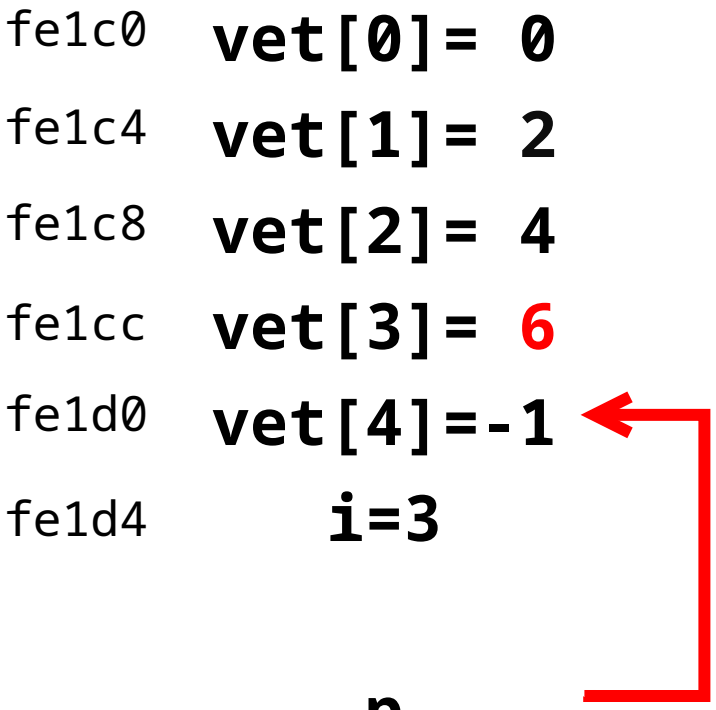
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 3

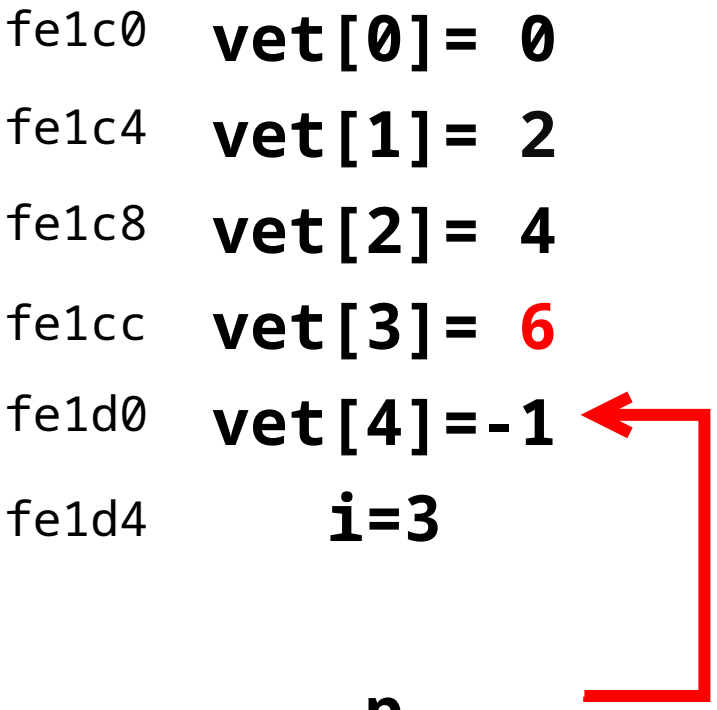
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 3

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]= 6
fe1d0   vet[4]=-1
fe1d4      i=4

           p
```

i = 4

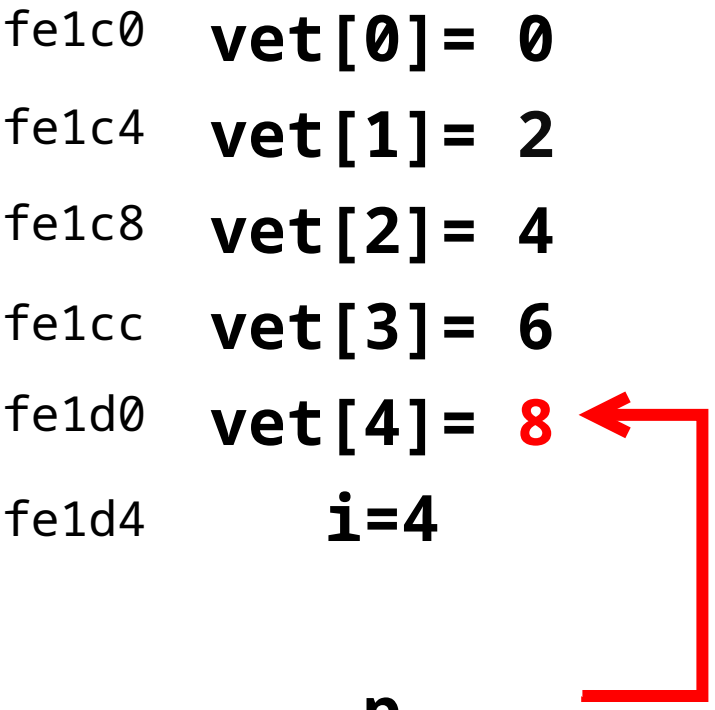
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7ffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7ffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7ffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7ffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7ffffffe1d4)
indirizzo di i: 0x7ffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 4

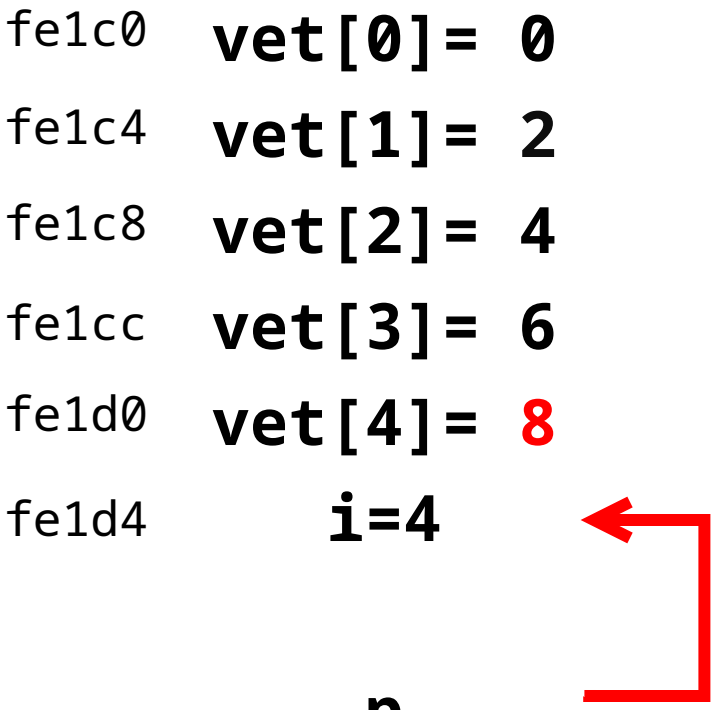
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 4

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(p++) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]= 6
fe1d0   vet[4]= 8
fe1d4      i=4


           p
```

i = 4

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0;
    *(p++)
    // [pri
}
```

Come mai non va in segmentation fault?

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]= 6
fe1d0   vet[4]= 8
fe1d4      i=4


           p
```

i = 4

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0;
    *(p++)
    // [pri
}
```

Come mai non va in segmentation fault?

Sto accedendo in lettura ad una cella dimensionata correttamente per un intero!

```
fe1c0   vet[0]= 0
fe1c4   vet[1]= 2
fe1c8   vet[2]= 4
fe1cc   vet[3]= 6
fe1d0   vet[4]= 8
fe1d4      i=4


           p
```

i = 4

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = 0, *p=-1, (p=0x7fffffffe1c4)
i = 1, vet[1] = 2, *p=-1, (p=0x7fffffffe1c8)
i = 2, vet[2] = 4, *p=-1, (p=0x7fffffffe1cc)
i = 3, vet[3] = 6, *p=-1, (p=0x7fffffffe1d0)
i = 4, vet[4] = 8, *p=4, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [0 2 4 6 8 ]
```

Arra

Modifi

Visuali

È sempre così??

NO! Questa era una semplificazione, chi dispone le variabili in memoria è il compilatore.

Possiamo solo limitarci ad osservare cosa succede, ed in più aggiungere una variabile o cambiarne l'ordine può cambiare tutto

```
fe1c0   v
fe1c4   v
fe1c8   v
fe1cc   v
fe1d0   v
fe1d4
```

```
valori di vet: [0 2 4 6 8 ]
```

Arra

Modifi

Visuali

N.B.: Anche la rappresentazione dello stack come nelle slide è una semplificazione di ciò che accade realmente.

Abituatevi comunque a schematizzarlo, per entrare meglio nell'ottica dei puntatori

```
fe1c0   v
fe1c4   v
fe1c8   v
fe1cc   v
fe1d0   v
fe1d4
```

```
valori di vet: [0 2 4 6 8 ]
```

Array e Puntatori

Modifica dei puntati

Modifichiamo nuovamente il programma … Cambiamo dove andiamo a scrivere all’interno del loop for: anziché usare **p++**, vediamo cosa succede con **++p**

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

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(p++) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nindirizzo di i: %p", &i);
    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

Ovvero cambio da:

```c
*(p++) = 2*i;
```

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

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(++p) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nindirizzo di i: %p", &i);
    printf("\nvalori di vet: [");
    for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

Ovvero cambio da:

```c
*(p++) = 2*i;
```

a:

```c
*(p++) = 2*i;
```

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

int main(){
    int vet[L] = {-1,-1,-1,-1,-1};
    int *p, i;

    p = vet;

    printf("\nvalori di vet: [");
    for(i = 0; i < L; i++) printf("%d ", vet[i]);
    printf("]");

    for(i = 0; i < L; i++){
        *(++p) = 2*i;
        printf("\ni = %d, vet[%d] = %d, *p=%d, (p=%p)", i, i, vet[i], *p, p);
    }
    printf("\nindirizzo di i: %p", &i);
    printf("\nvalori di vet: [");
    for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
    printf("]\n");
    return 0;
}
```

Stampiamo anche “un po’ più in là” per capire meglio cosa succede

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

1. Prima incremento p
2. Scrivo nella cella puntata da p il valore 2*i

- La prima volta, incremento scriverò in vet[1]=0

Array e Puntatori

Modifica dei puntati

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
```

Concentri

1. Prima
2. Scrivo

- La pri
  vet[1]

Modifica dei puntati

Si consideri il seguente programma:

Copio nel puntatore l'indirizzo di `vet[0]`.

Coincide con

`p = &vet[0]`

```
int vet[L] = {-1,-1,-1,-1,-1}
int *p, i;
p = vet;

// [print valori di vet]

for(i = 0; i < L; i++){
  *(p + i) = 2*i;
  // [print i, vet[i], *p, p]

}
```

[Dallo screenshot della slide in cui spiegavamo il programma]
p era inizializzato con vet, ≡ p=&vet[0]

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

1. Prima incremento p
2. Scrivo nella cella puntata da p il valore 2*i

- La prima volta, incremento scriverò in vet[1]=0
- La seconda, incrementoe scrivo in vet[2]=2
- […]

- E l’ultima? Dopo aver incrementato p, a cosa punterà?

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Concentriamoci sul for ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
}
```

Cerchiamo di analizzare lo stack come prima, per capire meglio (sempre con la nostra semplificazione, per rimanere coerenti con le slides prima)

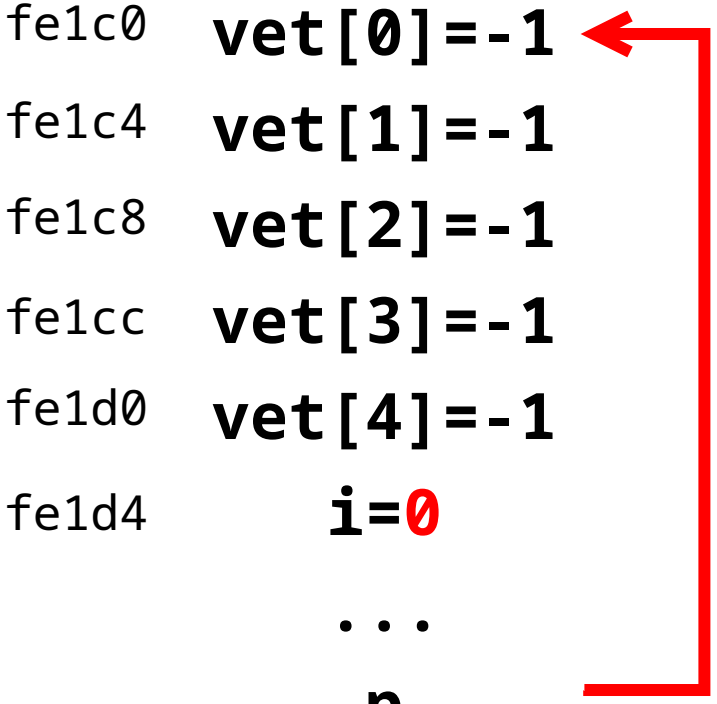
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

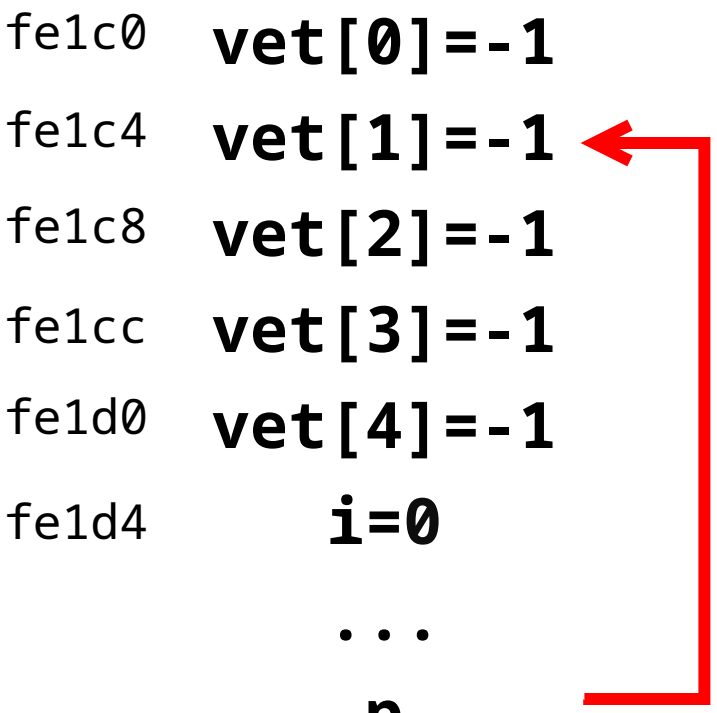
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

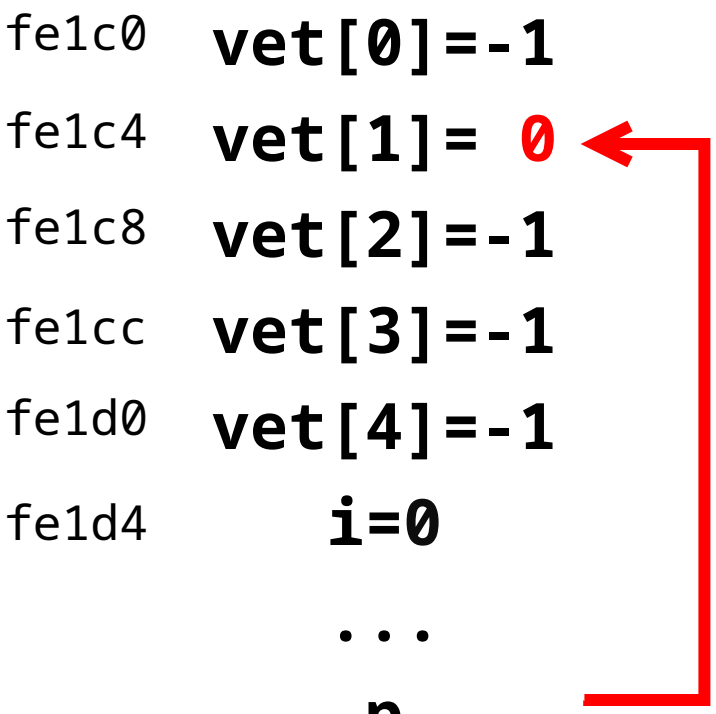
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

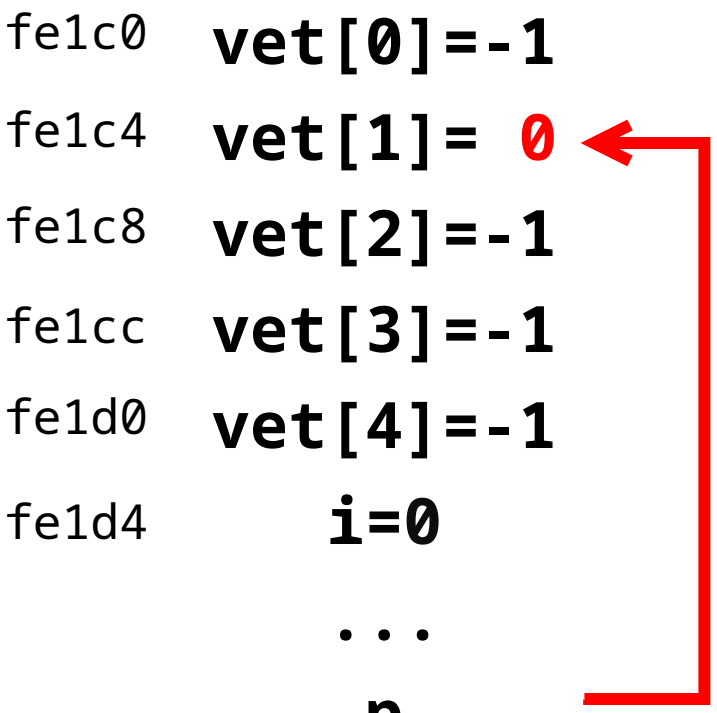
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 0

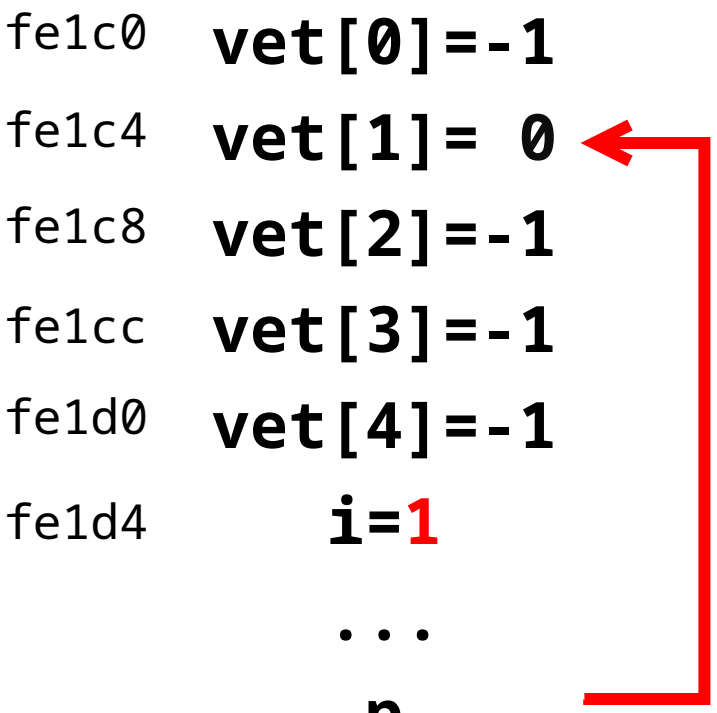
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

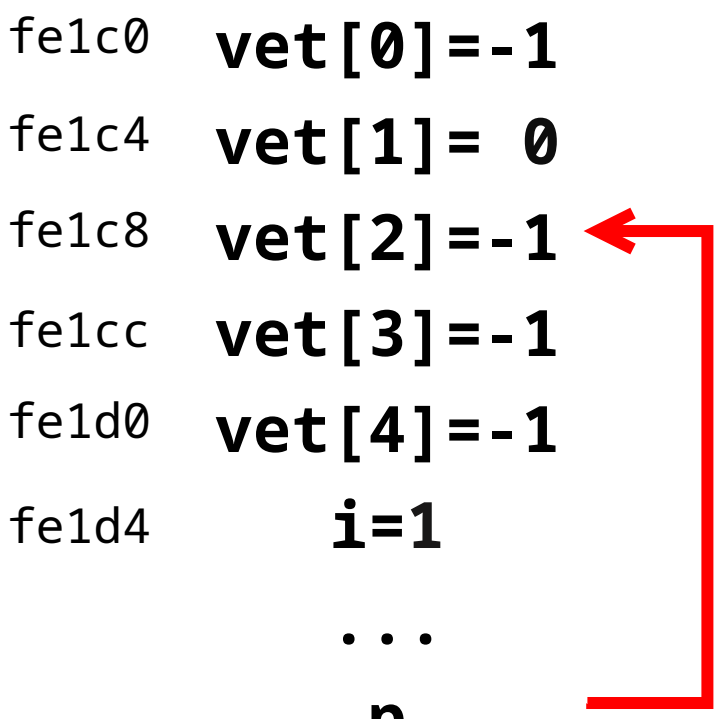
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

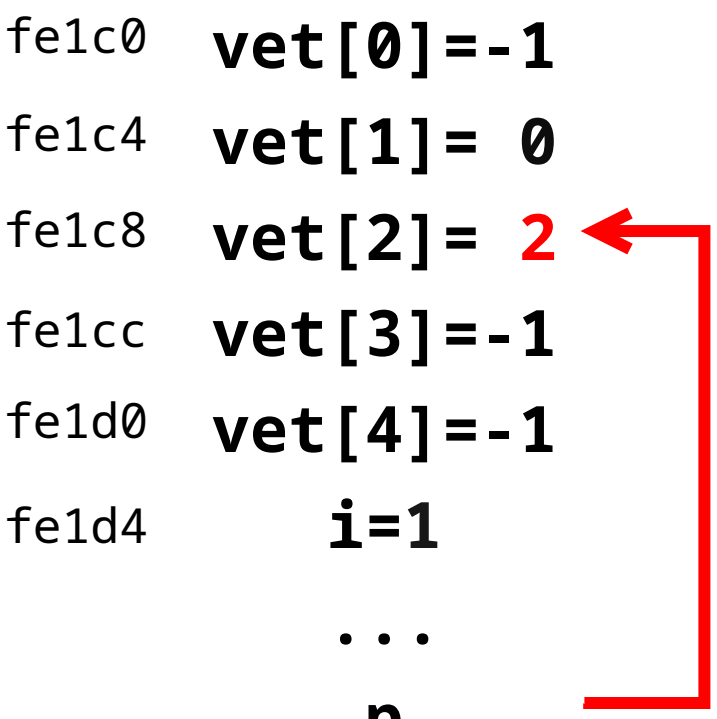
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

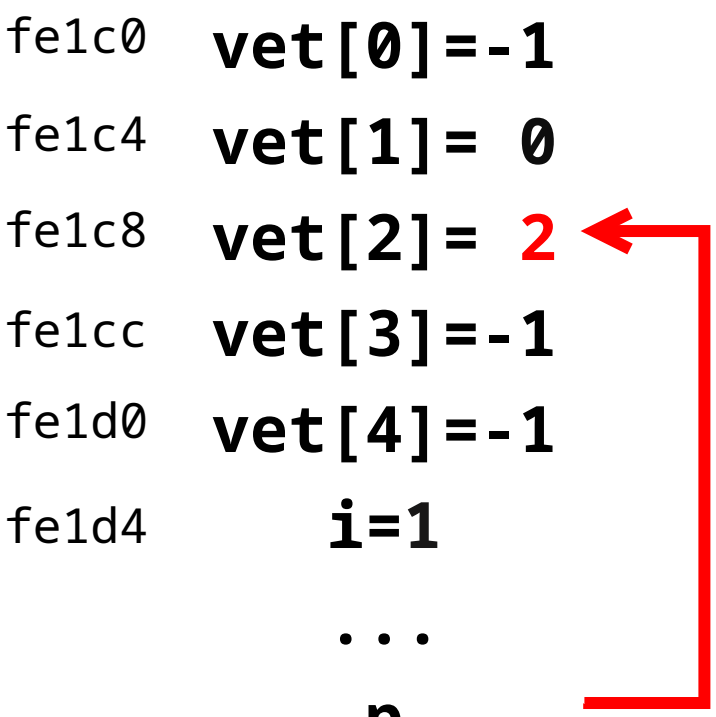
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 1

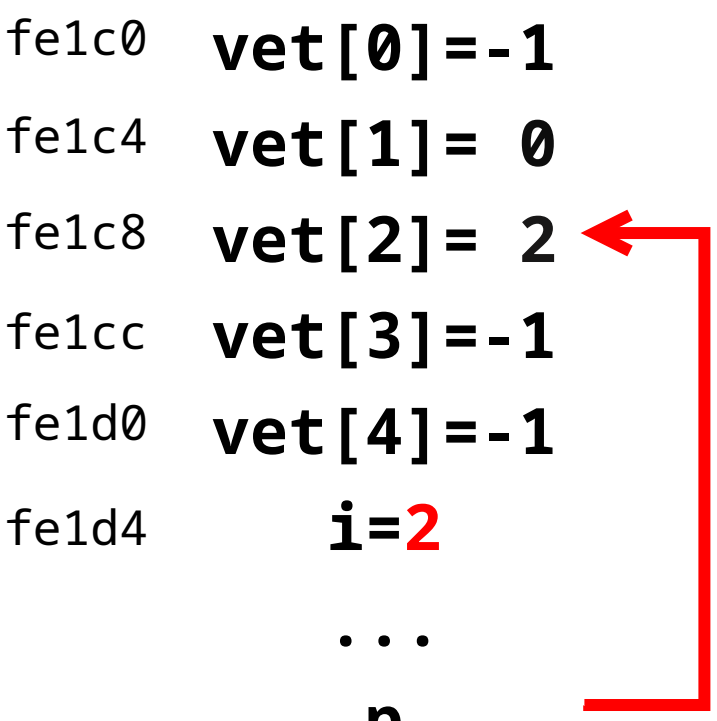
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 2

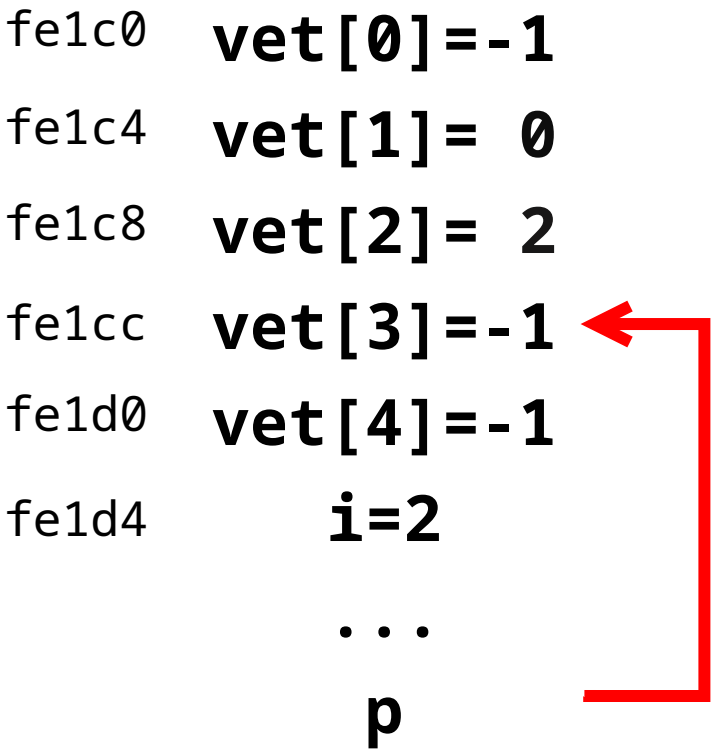
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 2

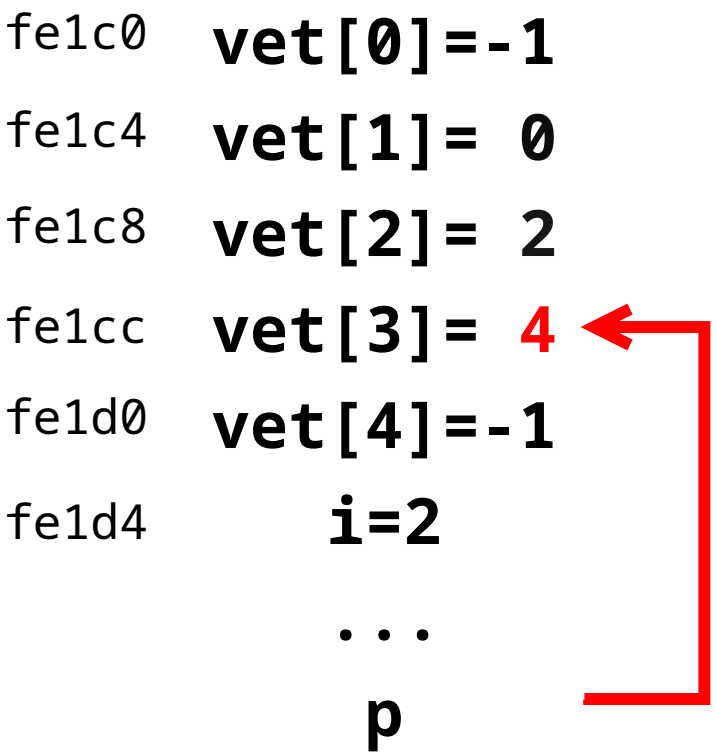
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 2

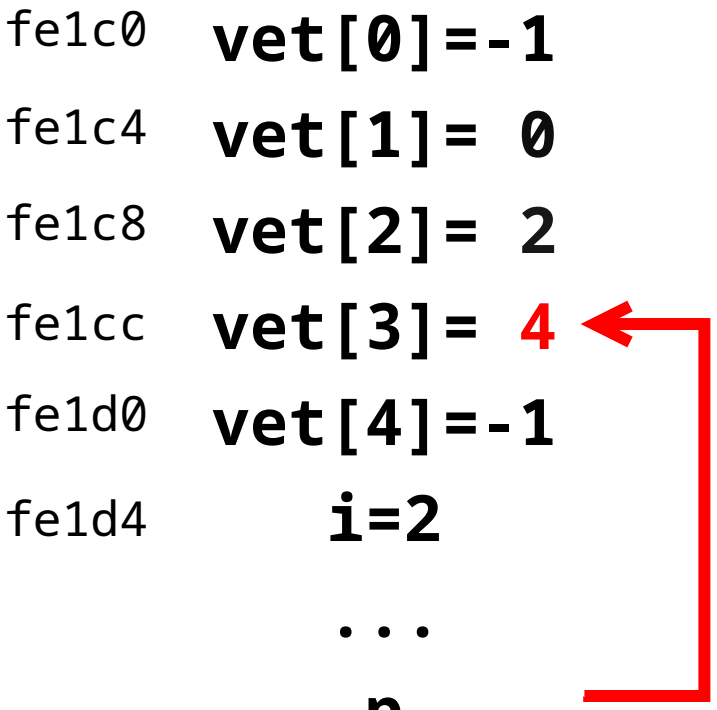
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 2

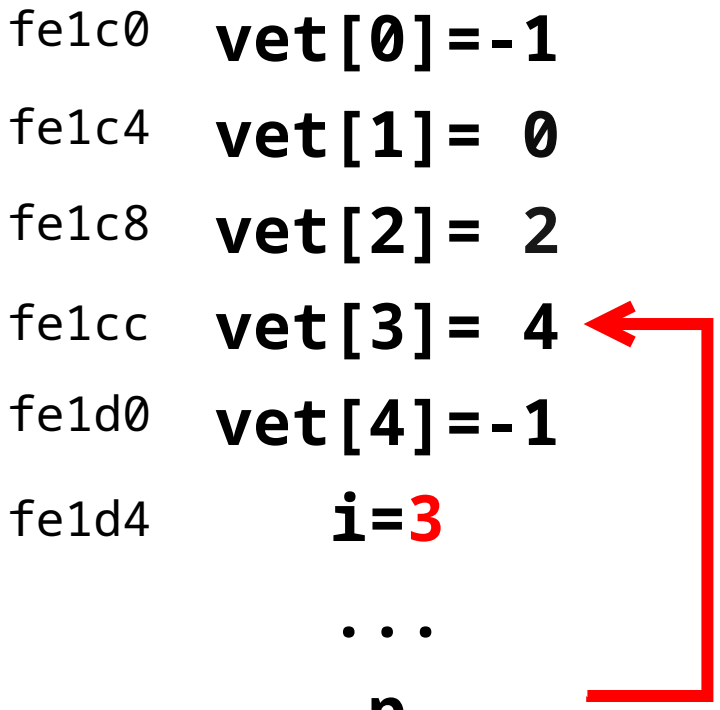
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 3

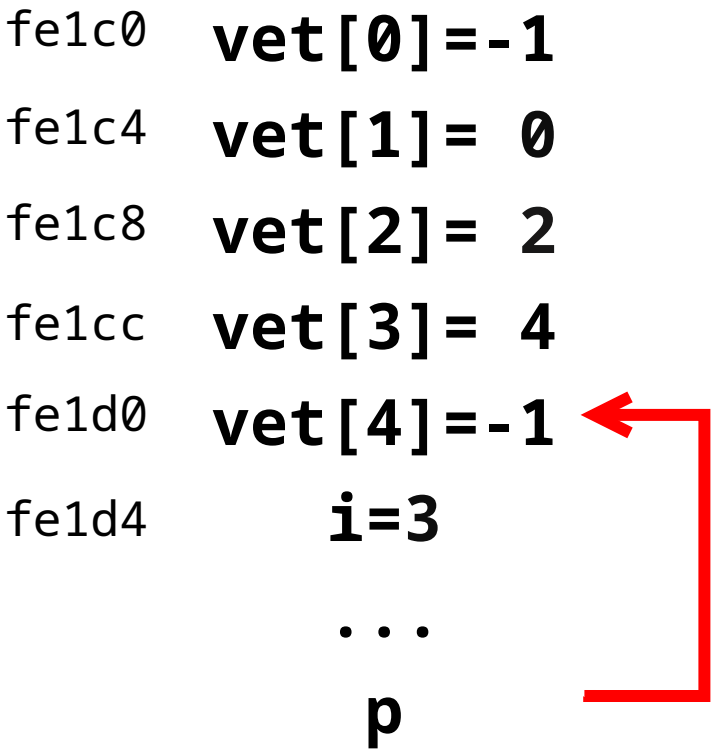
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 3

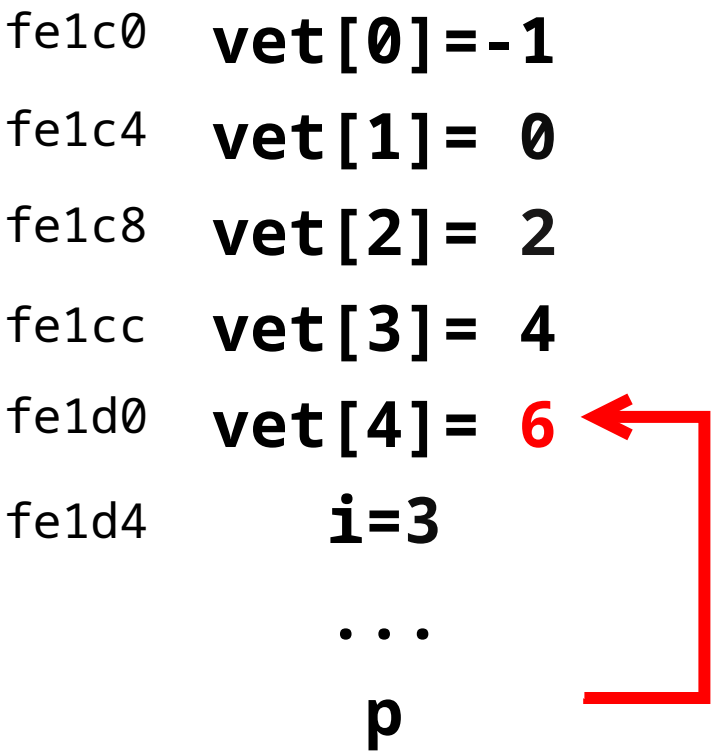
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 3

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]=-1
fe1c4   vet[1]= 0
fe1c8   vet[2]= 2
fe1cc   vet[3]= 4
fe1d0   vet[4]= 6
fe1d4      i=3
           ...
            p
```

i = 3

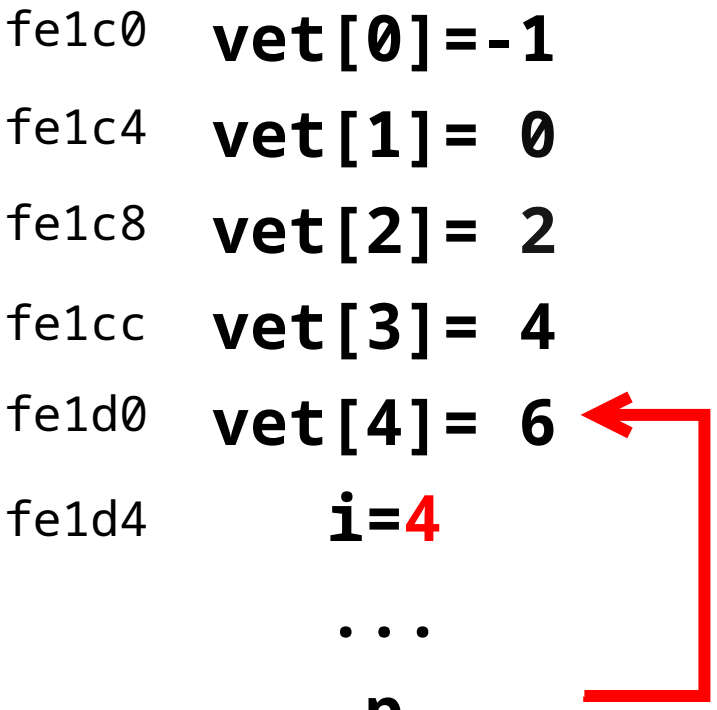
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 4

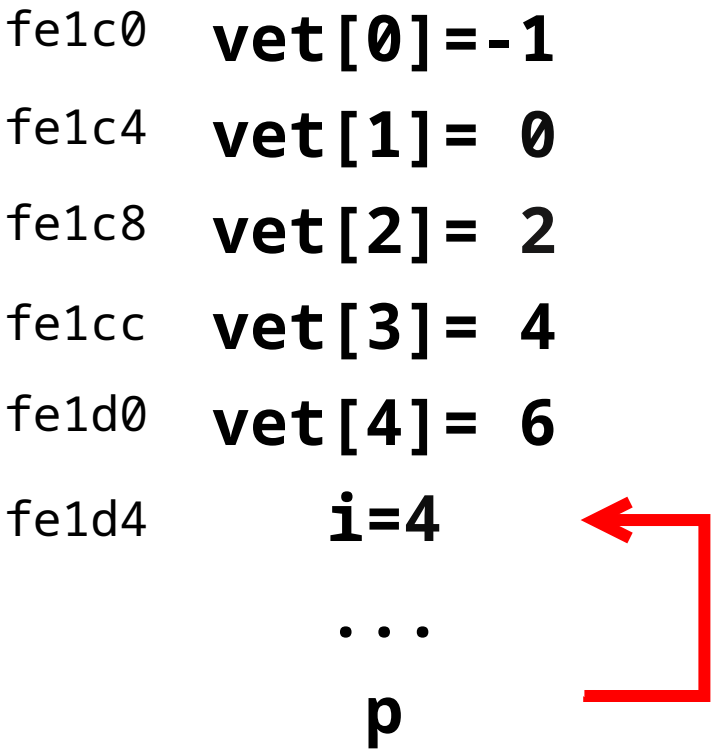
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 4

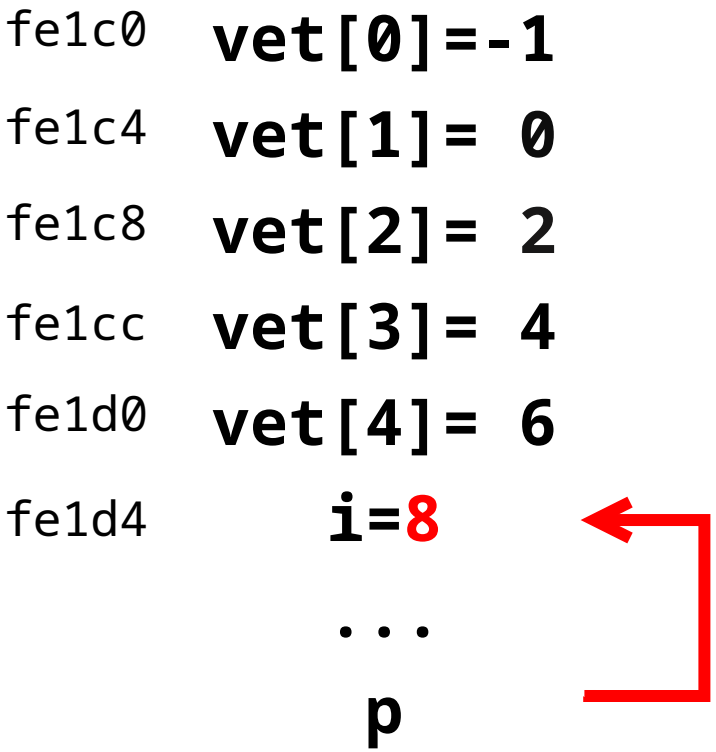
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack …

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 8

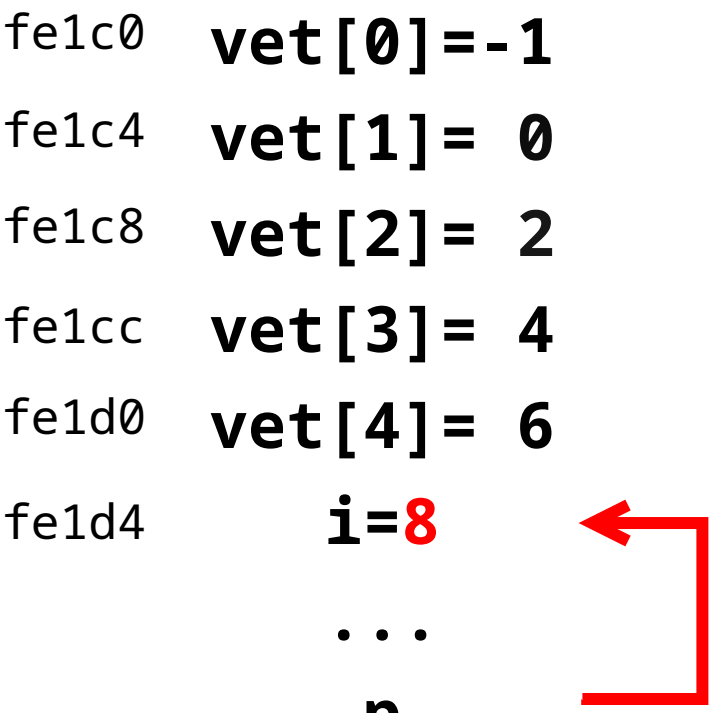
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 8

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]=-1
fe1c4   vet[1]= 0
fe1c8   vet[2]= 2
fe1cc   vet[3]= 4
fe1d0   vet[4]= 6
fe1d4      i=8
           ...
            p
```

i = 8

vet[8]

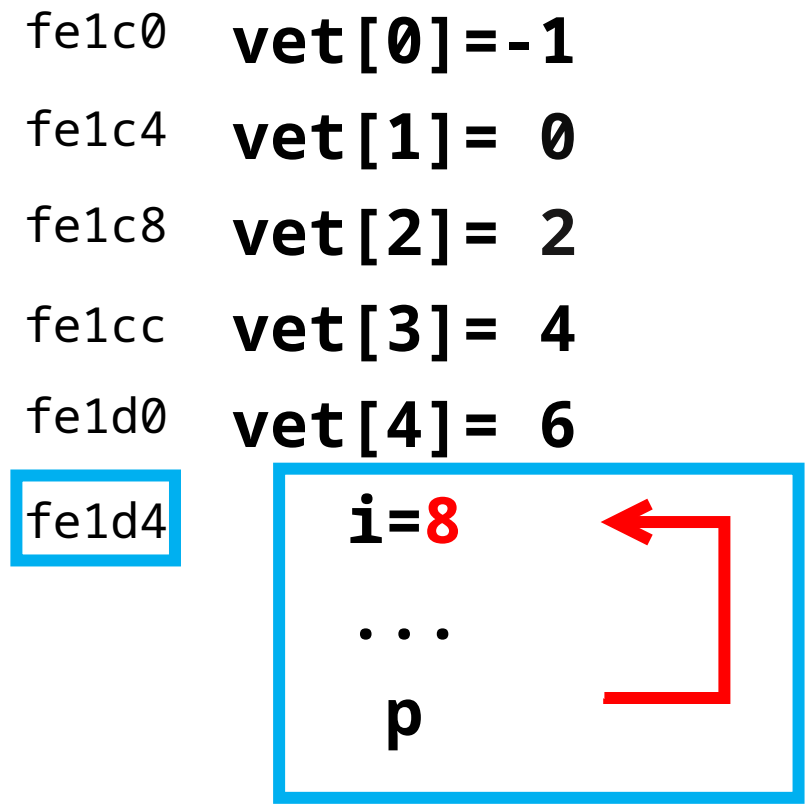
```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```



i = 8

p “punta” i

```
ri di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack ...

```
for(i = 0; i < L; i++){
    *(++p) = 2*i;
    // [print i, vet[i], *p, p]
}
```

```
fe1c0   vet[0]=-1
fe1c4   vet[1]= 0
fe1c8   vet[2]= 2
fe1cc   vet[3]= 4
fe1d0   vet[4]= 6
fe1d4      i=8
           ...
            p
```

i = 8

Perché allora qui otteniamo 5 se abbiamo detto che sovrascriviamo i??

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Visualizziamo lo stack

```
for(i = 0;
    *(++p)
    // [pri
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

Hint su un pezzo di codice che viene dopo il loop appena analizzato ...

```
fe1c0   vet[0]=-1
fe1c4   vet[1]= 0
fe1c8   vet[2]= 2
fe1cc   vet[3]= 4
fe1d0   vet[4]= 6
fe1d4      i=8
           ...
            p
```

i = 8

Perché allora qui otteniamo 5 se abbiamo detto che sovrascriviamo i??

```
valori di vet: [-1 -1 -1 -1 -1 ]
i = 0, vet[0] = -1, *p=0, (p=0x7fffffffe1c4)
i = 1, vet[1] = 0, *p=2, (p=0x7fffffffe1c8)
i = 2, vet[2] = 2, *p=4, (p=0x7fffffffe1cc)
i = 3, vet[3] = 4, *p=6, (p=0x7fffffffe1d0)
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

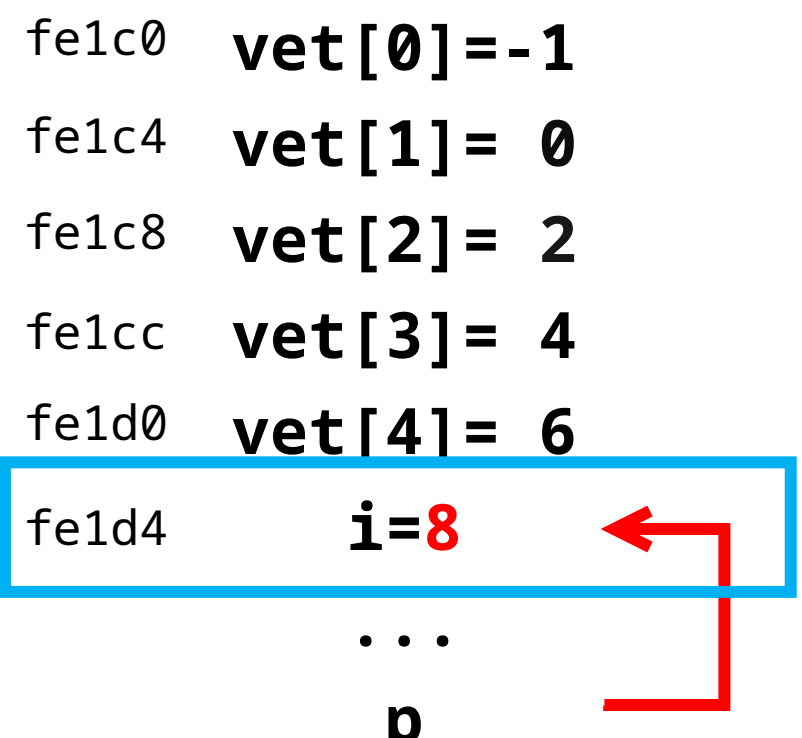
Modifica dei puntati

Visualizziamo lo stack

```
for(i = 0;
    *(++p)
    // [pri
```

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

Hint su un pezzo di codice che viene dopo il loop appena analizzato ...



i = 8

È comunque la cella riservata ad i ... Anche nel loop dopo! Quando i vale 5, vet[5] stampa i (quindi 5). Dopo, i continuerà a crescere per stampare gli altri valori

```
valori di v
i = 0, vet[
i = 1, vet[
i = 2, vet[
i = 3, vet[
i = 8, vet[8] = -7552, *p=8, (p=0x7fffffffe1d4)
indirizzo di i: 0x7fffffffe1d4
valori di vet: [-1 0 2 4 6 5 -1620749056 -
865439461 -7552 32767 ]
```

Array e Puntatori

Modifica dei puntati

Come mai non otteniamo segmentation fault?

Array e Puntatori

Modifica dei puntati

Come mai non otteniamo segmentation fault?

Per come è stata disposta la memoria (ricorda, aggiungere un’altra variabile o cambiare compilatore potrebbe portare ad un risultato diverso), lì posso accedere, ma **non è garantito!**

Array e Puntatori

Modifica dei puntati

Trasformando il ciclo di stampa da:

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

Array e Puntatori

Modifica dei puntati

Trasformando il ciclo di stampa da:

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

a:

```
for(i = 0; i < L+10000; i++) printf("%d ", vet[i]);
```

Array e Puntatori

Modifica dei puntati

Trasformando il ciclo di stampa da:

```
for(i = 0; i < L+5; i++) printf("%d ", vet[i]);
```

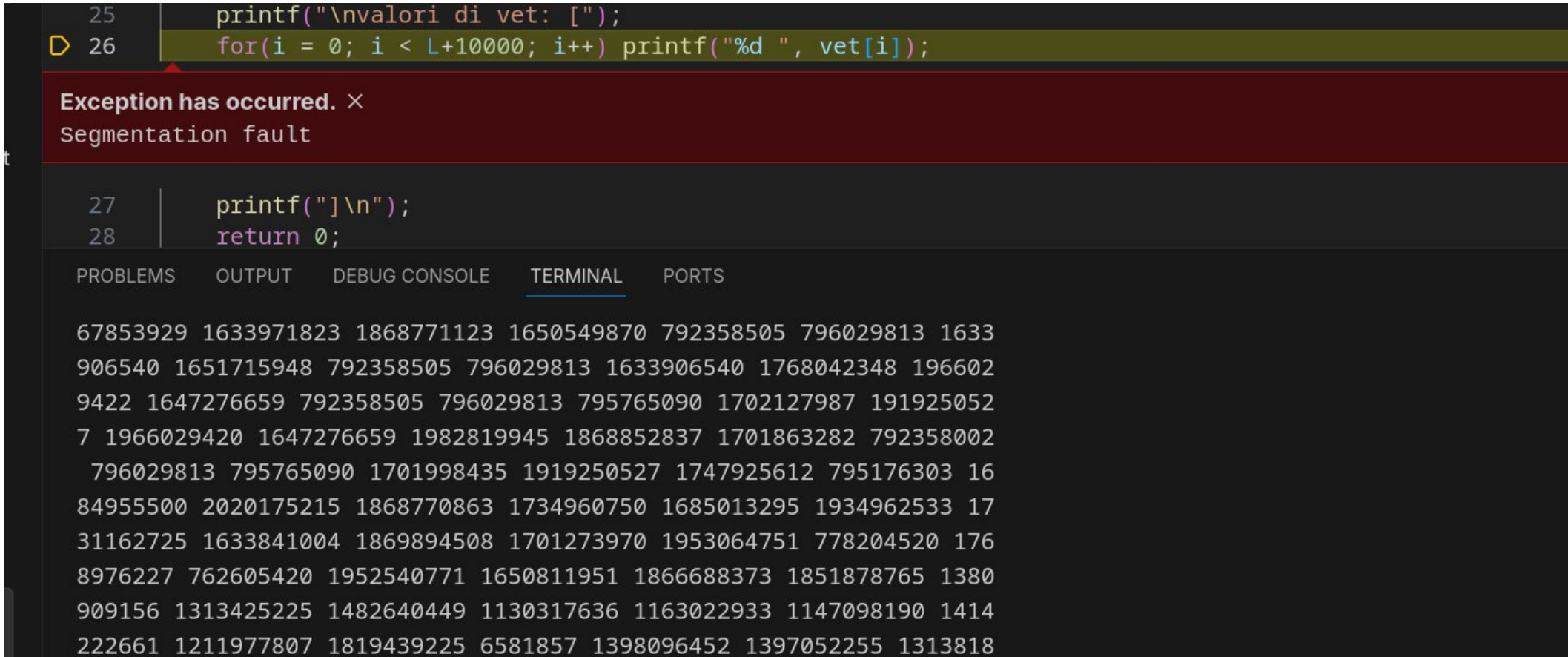
a:

```
for(i = 0; i < L+10000; i++) printf("%d ", vet[i]);
```

Array e Puntatori

Modifica dei puntati

Ho segmentation fault, vado ad accedere a dove non dovrei!



Thank You!

Mail: luca1.alessandrini@polimi.it

Website: https://alessandriniluca.github.io/