Allocazione dinamica

Fondamenti di Informatica, AA 2022/23

Luca Cassano

luca.cassano@polimi.it

Allocazione dinamica

- In C dobbiamo dichiarare tutte le variabili necessarie in fase di progetto mentre scriviamo il codice del programma...

Allocazione dinamica

- In C dobbiamo dichiarare tutte le variabili necessarie in fase di progetto mentre scriviamo il codice del programma...
  - ... tuttavia a volte sarebbe comodo avere uno strumento più flessibile della dichiarazione di una variabile per creare questo spazio di memoria

Allocazione dinamica

- In C dobbiamo dichiarare tutte le variabili necessarie in fase di progetto mentre scriviamo il codice del programma...
  - ... tuttavia a volte sarebbe comodo avere uno strumento più flessibile della dichiarazione di una variabile per creare questo spazio di memoria
  - non sempre siamo in grado di prevedere a priori quanti dati dobbiamo memorizzare

Allocazione dinamica

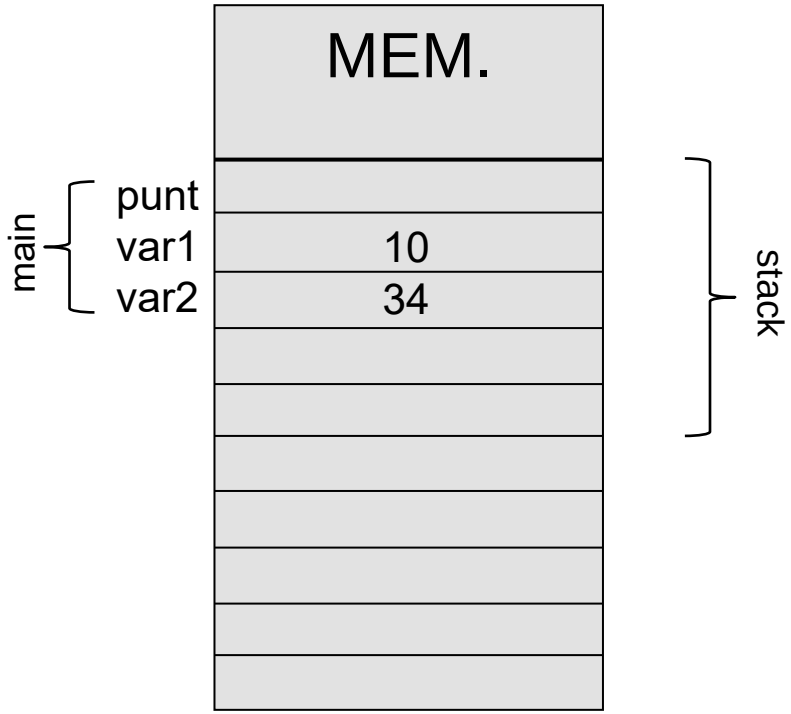
- In C dobbiamo dichiarare tutte le variabili necessarie in fase di progetto mentre scriviamo il codice del programma...
  - ... tuttavia a volte sarebbe comodo avere uno strumento più flessibile della dichiarazione di una variabile per creare questo spazio di memoria
  - non sempre siamo in grado di prevedere a priori quanti dati dobbiamo memorizzare
  - la quantità di informazioni da memorizzare può cambiare durante l'esecuzione del programma

Allocazione dinamica

- In C dobbiamo dichiarare tutte le variabili necessarie in fase di progetto mentre scriviamo il codice del programma...
  - ... tuttavia a volte sarebbe comodo avere uno strumento più flessibile della dichiarazione di una variabile per creare questo spazio di memoria
  - non sempre siamo in grado di prevedere a priori quanti dati dobbiamo memorizzare
  - la quantità di informazioni da memorizzare può cambiare durante l'esecuzione del programma
  - in un sottoprogramma, vogliamo poter creare uno spazio di memoria (ed assegnarli un valore) che sopravviva alla conclusione del sottoprogramma stesso

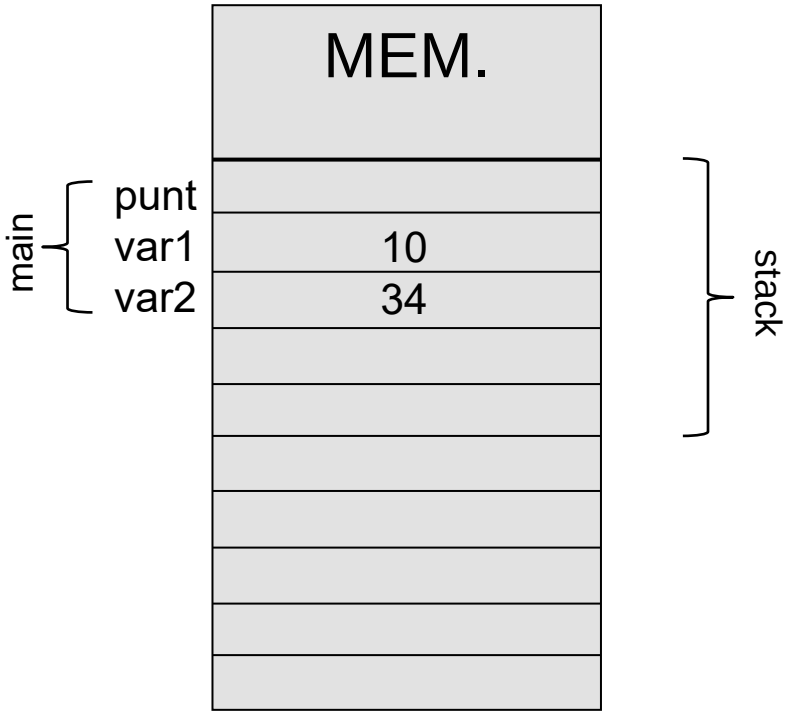
Allocazione dinamica stack e heap

- Finora abbiamo visto che la memoria di un programma in esecuzione è gestita tramite una pila (stack)



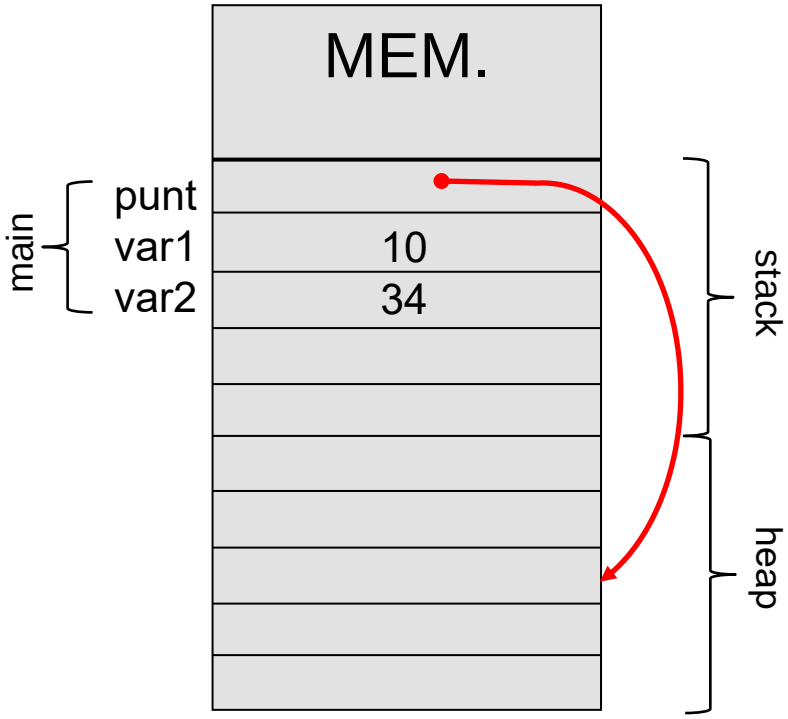
Allocazione dinamica stack e heap

- Finora abbiamo visto che la memoria di un programma in esecuzione è gestita tramite una pila (stack)
  - Un record di attivazione viene aggiunto quando viene invocata un sottoprogramma e tolto quando il sottoprogramma termina (politica LIFO)
  - Il record di attivazione contiene tutte le variabili dichiarate nel sottoprogramma



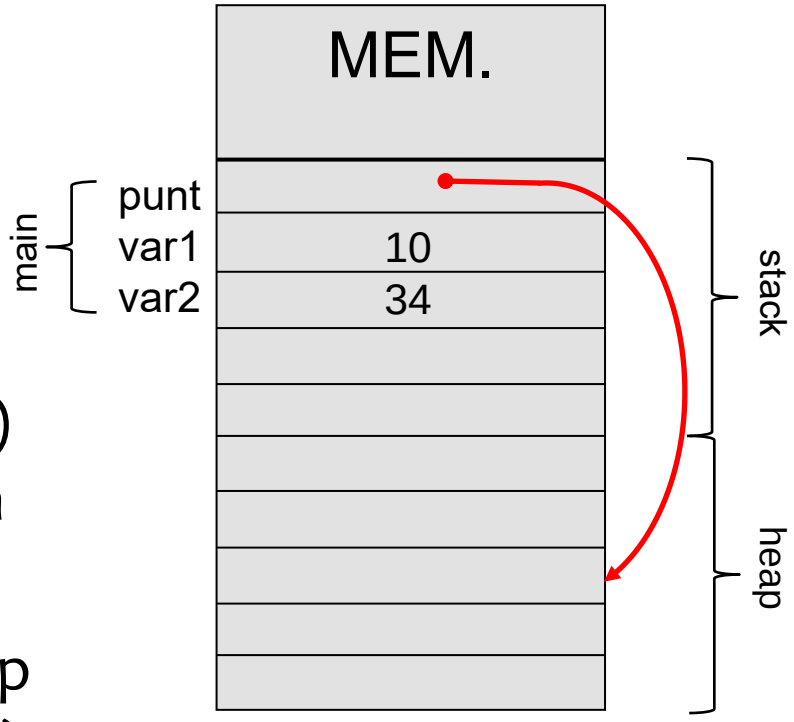
Allocazione dinamica stack e heap

- La memoria riservata ad un programma in esecuzione contiene un'altra area detta heap in cui è possibile allocare dinamicamente dello spazio



Allocazione dinamica stack e heap

- La memoria riservata ad un programma in esecuzione contiene un'altra area detta heap in cui è possibile allocare dinamicamente dello spazio
  - Usiamo i puntatori dichiarati nel codice (che si trovano nella stack) per puntare alla memoria allocata dinamicamente nello heap
  - la politica di allocazione nello heap non è predefinito (heap=mucchio)



Allocazione dinamica – malloc()

- In C è possibile allocare dinamicamente uno spazio di memoria tramite il sottoprogramma `malloc` contenuta nella libreria `stdlib.h`:

```
tipo *p = malloc (size);
```

- `size` è la quantità (in byte) di memoria da allocare ed è in genere espressa come `n*sizeof(tipo)`
  - Cioè un array di `n` elementi

Allocazione dinamica – malloc()

- Il sottoprogramma riserva uno spazio di memoria nello heap delle dimensioni richieste e ne restituisce l'indirizzo
  - `malloc` restituisce un valore `void*` cioè un indirizzo senza tipo
  - Viene eseguito quindi un cast implicito (qualcuno mette l'operatore di cast)
- Nel caso non sia possibile riservare uno spazio di memoria con le caratteristiche richieste, il sottoprogramma ritorna `NULL`

Allocazione dinamica – free()

- In C è necessario deallocare (liberare) uno spazio di memoria non più necessario con il sottoprogramma `free` contenuta nella libreria `stdlib.h`:

```
free(indirizzo);
```

- Il sottoprogramma riceve come parametro in ingresso l'indirizzo di uno spazio di memoria precedentemente allocato e lo libera (rendendolo riutilizzabile in seguito)
- Attenzione a
  - Non liberare uno spazio di memoria ancora in uso
  - Non usare uno spazio di memoria dopo averlo liberato!
  - Non dimenticarsi di deallocare la memoria non più usata

Allocazione dinamica – allocazione di array

```
#include<stdio.h>
#include<stdlib.h>
#define N 5
int main(){
  int *p, i;
  p=malloc(sizeof(int)*N);
  if(p != NULL){
    for(i=0;i<N;i++)
      scanf("%d",(p+i));
      /*...*/
    free(p);
  } else
    printf("Err. alloc.");
  return 0;
}
```

Dichiarazione di una variabile puntatore

Allocazione dinamica della memoria

Controllo della corretta allocazione della memoria

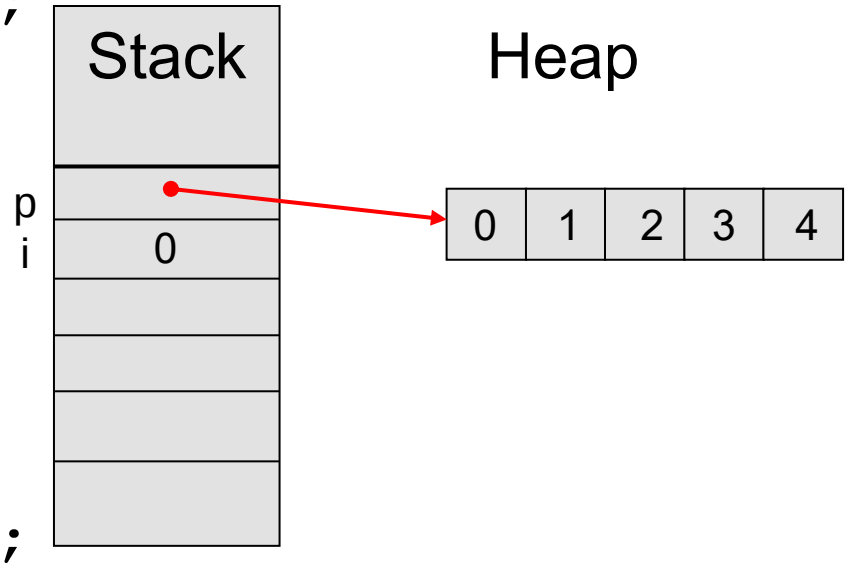
Utilizzo della memoria

Deallocazione della memoria

Gestione degli errori

Allocazione dinamica – allocazione di array

```
#include<stdio.h>
#include<stdlib.h>
#define N 5
int main(){
  int *p, i;
  p=malloc(sizeof(int)*N);
  if(p != NULL){
    for(i=0;i<N;i++)
      scanf("%d",(p+i));
      /*...*/
    free(p);
  } else
    printf("Err. alloc.");
  return 0;
}
```



Allocazione dinamica – allocazione di matrici

- Si possono anche allocare dinamicamente delle matrici

```
#define R 5
#define C 5
int main(){
  int **m, i, j;
  m=malloc(sizeof(int*)*R);
  if(m != NULL){
    for(i=0;i<R;i++)
      *(m+i)=malloc(sizeof(int)*C);
      if(*(m+i) != NULL){
        for(j=0;j<C;j++)
          *(*(m+i)+j)=0;
  /*...*/
```

Dichiarazione di un doppio puntatore per puntare alla matrice

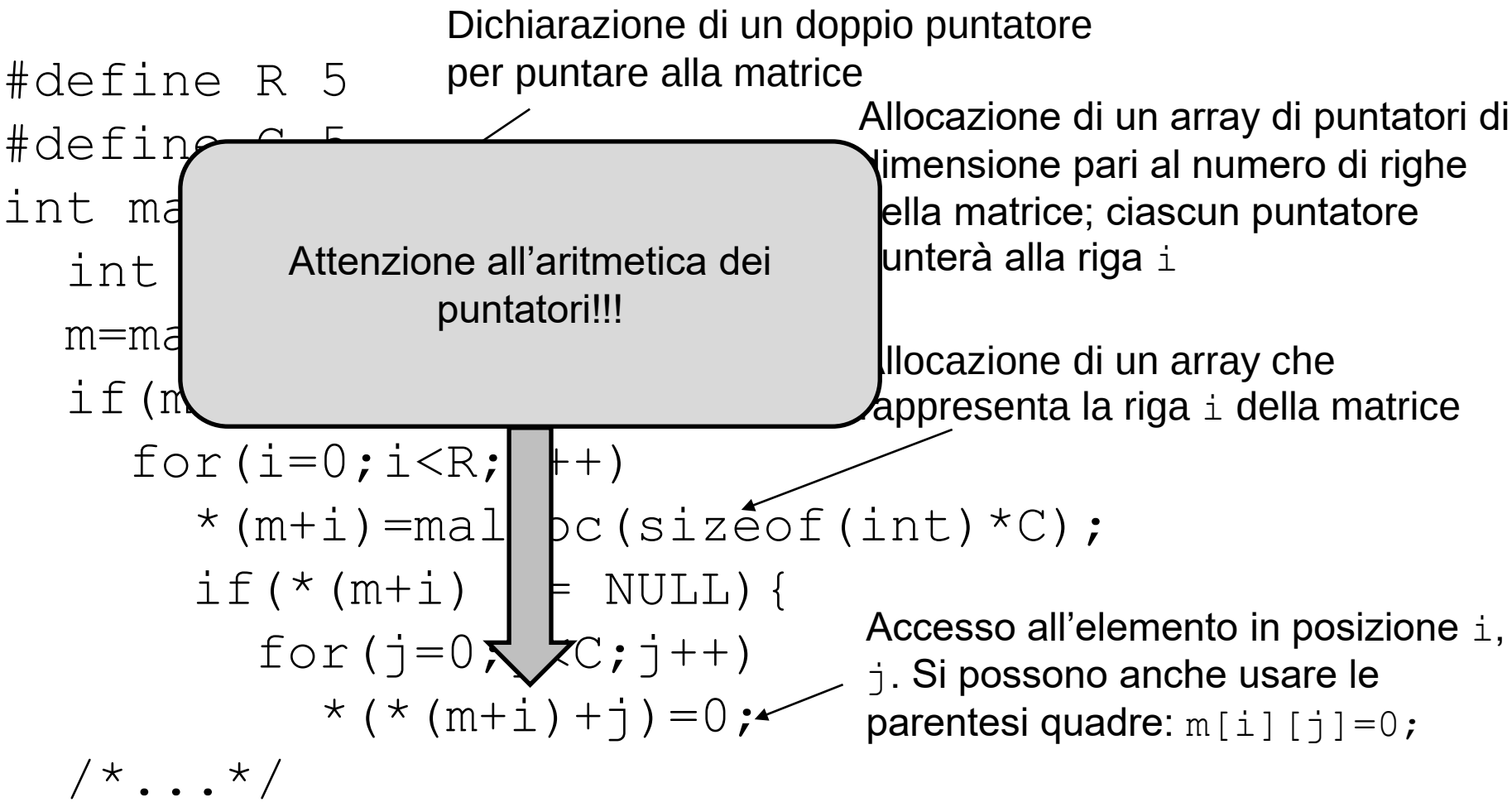
Allocazione di un array di puntatori di dimensione pari al numero di righe della matrice; ciascun puntatore punterà alla riga `i`

Allocazione di un array che rappresenta la riga `i` della matrice

Accesso all'elemento in posizione `i`, `j`. Si possono anche usare le parentesi quadre: `m[i][j]=0;`

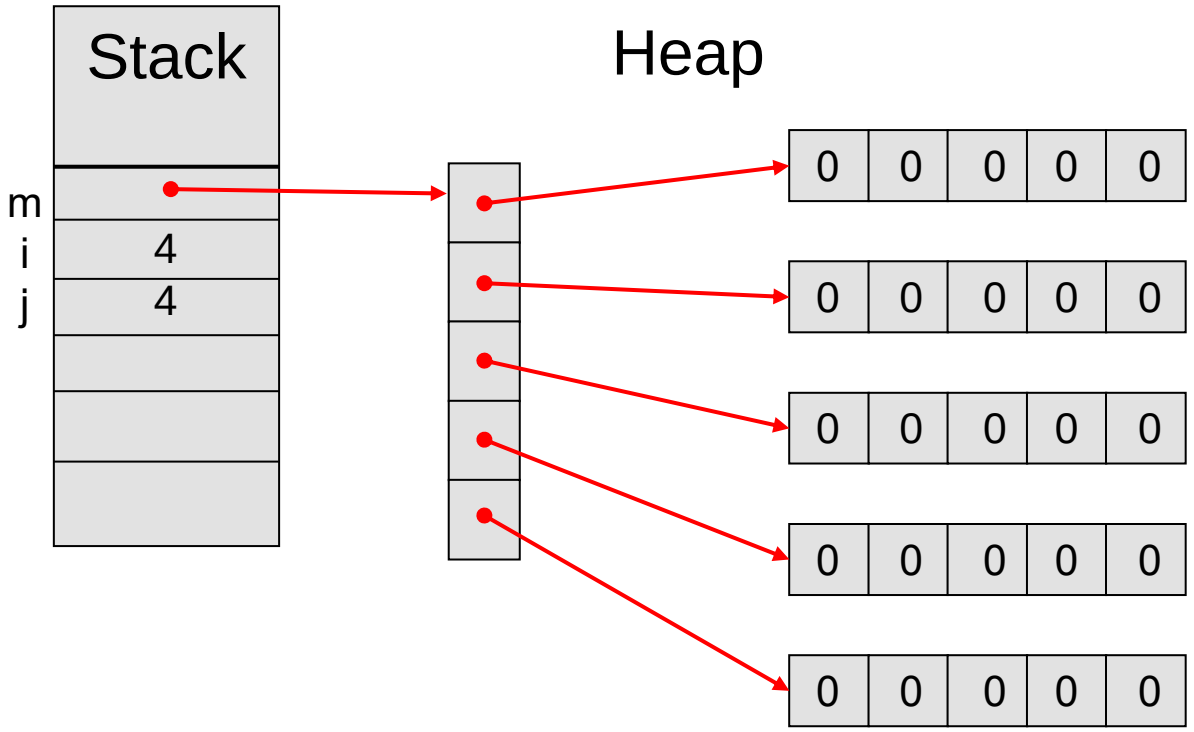
Allocazione dinamica – allocazione di matrici

- Si possono anche allocare dinamicamente delle matrici



Allocazione dinamica – allocazione di matrici

- Risultato dell'esecuzione del frammento di codice:



Allocazione dinamica – deallocazione di matrici

- Per liberare la memoria allocata per una matrice

```
#define R 5
#define C 5

int main(){
   int **m, i, j;

   /*...*/
   for(i=0;i<R;i++)
      free(*(m+i));
   free(m);
   /*...*/
```

Prima si deallocano i singoli array-riga

Poi si dealloca l'array di puntatori

Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

```
int* foo(){
  int *p;
  p=malloc(sizeof(int)*N);
  ...
  return p;
}

int main(){
  int *a;
  a=foo();
  ...
```

Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

```
int* foo(){
   int *p;
   p=malloc(sizeof(int)*N);
   ...
   return p;
}

int main(){
   int *a;
   a=foo();
   ...
```

foo()alloca dinamicamente la memoria

foo() restituisce l'indirizzo di memoria allocata

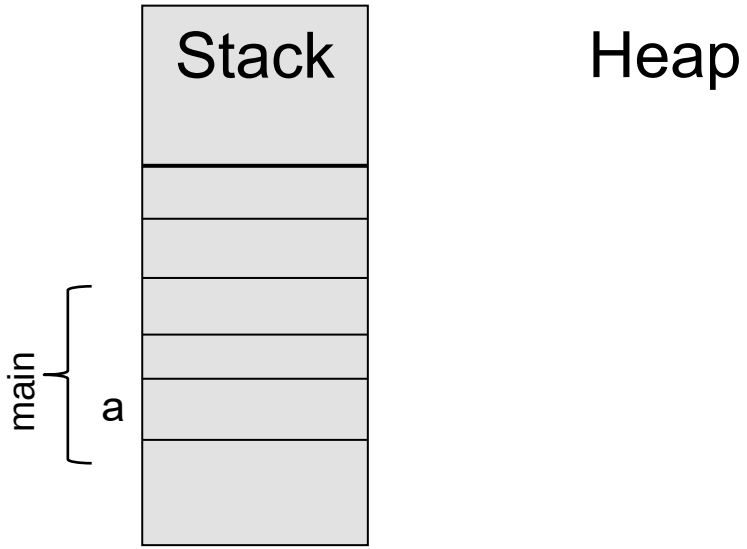
Il record di attivazione di foo() viene distrutto mentre la memoria allocata no!

Il `main()` salva l'indirizzo della memoria in un puntatore e la può quindi usare dopo la terminazione di `foo()`

Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

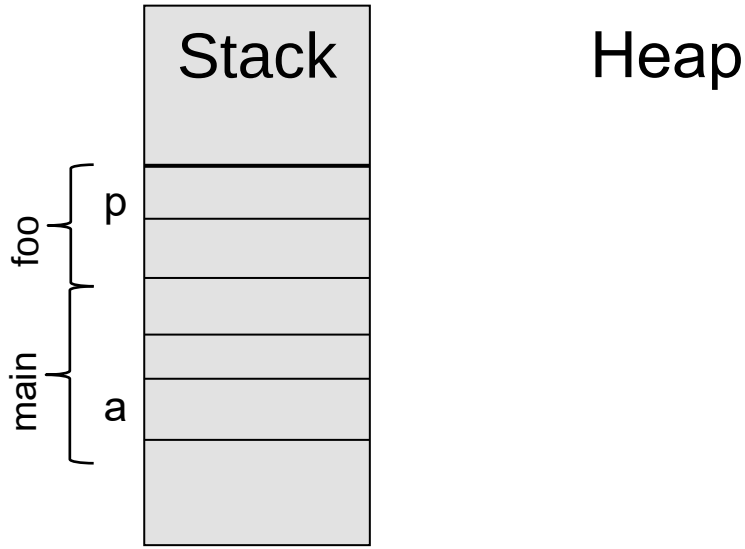
```
int* foo(){
   int *p;
   p=malloc(sizeof(int)*N);
   ...
   return p;
}

int main(){
   int *a;
   a=foo();
   ...
```



Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

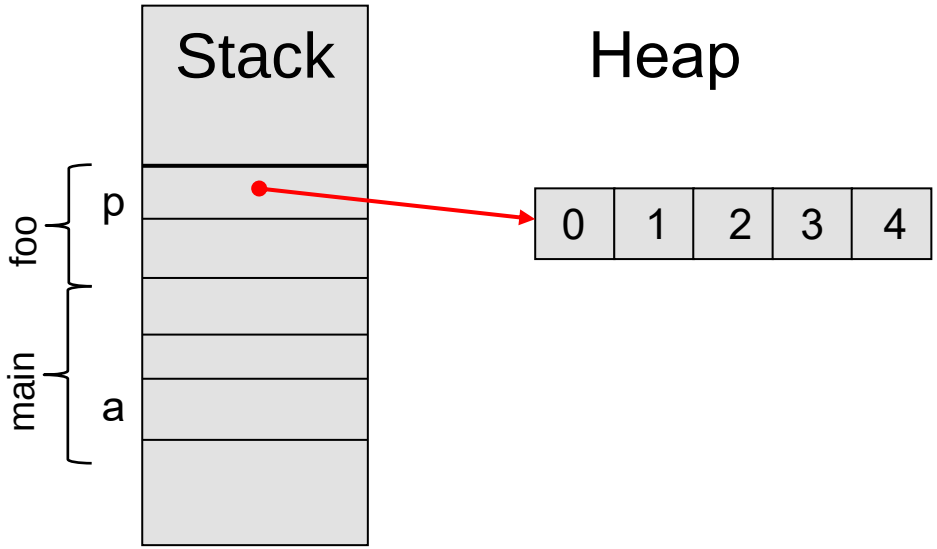
```
int* foo(){
   int *p;
   p=malloc(sizeof(int)*N);
   ...
   return p;
}

int main(){
   int *a;
   a=foo();
   ...
```



Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

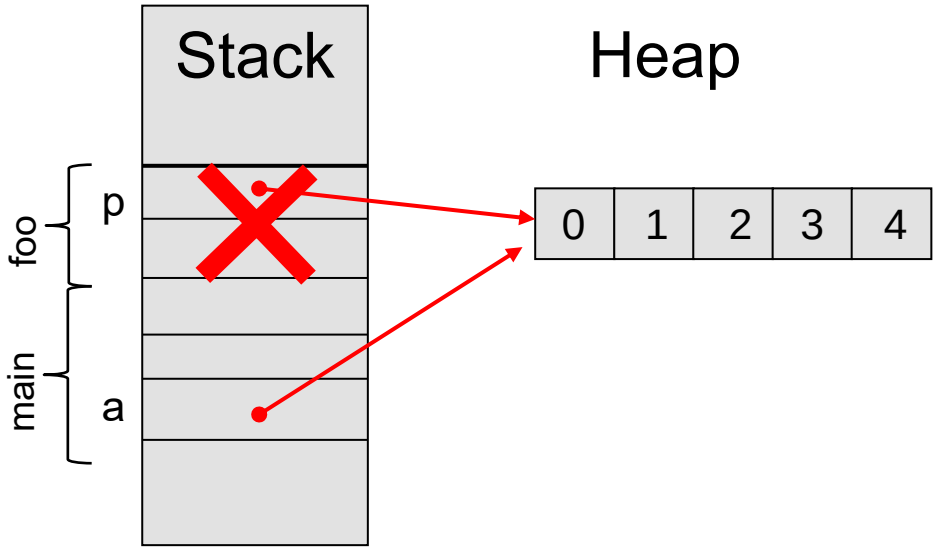
```
int* foo(){
   int *p;
   p=malloc(sizeof(int)*N);
   ...
   return p;
}

int main(){
   int *a;
   a=foo();
   ...
```



Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

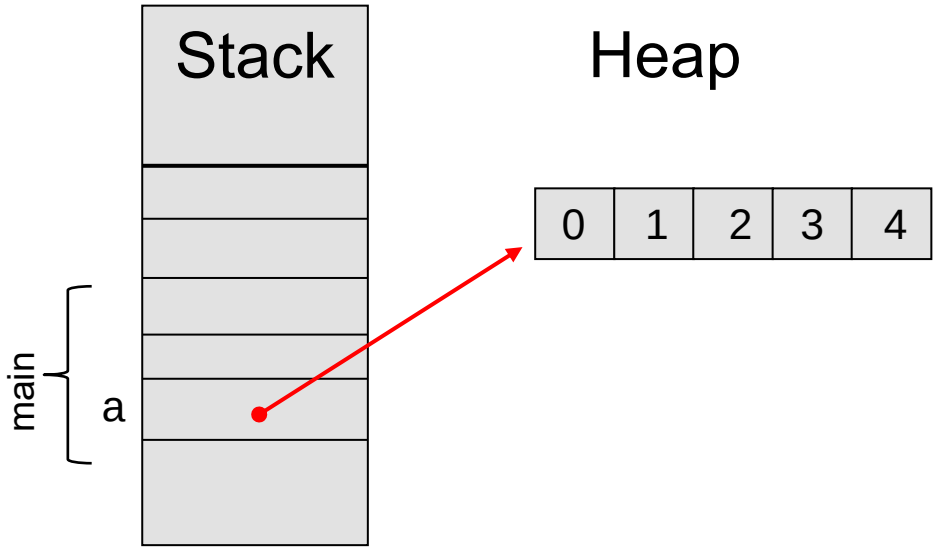
```
int* foo(){
   int *p;
   p=malloc(sizeof(int)*N);
   ...
   return p;
}

int main(){
   int *a;
   a=foo();
   ...
```



Allocazione dinamica – sottoprogrammi

- Un sottoprogramma può restituire della memoria allocata dinamicamente

```
int* foo(){
   int *p;
   p=malloc(sizeof(int)*N);
   ...
   return p;
}

int main(){
   int *a;
   a=foo();
   ...
```



Allocazione dinamica – sottoprogrammi

- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

Allocazione dinamica – sottoprogrammi

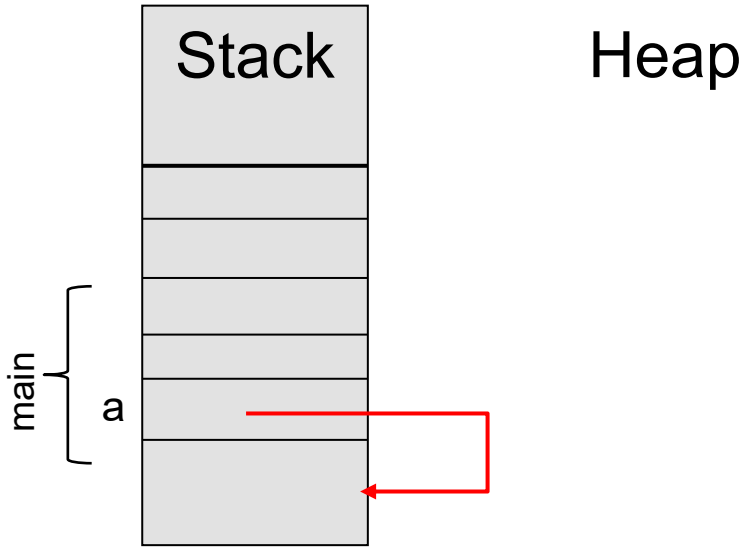
- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

```
void foo(int *p){
  p=malloc(sizeof(int)*N);
}

int main(){
  int *a;
  foo(a);
  ...
```

Allocazione dinamica – sottoprogrammi

- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

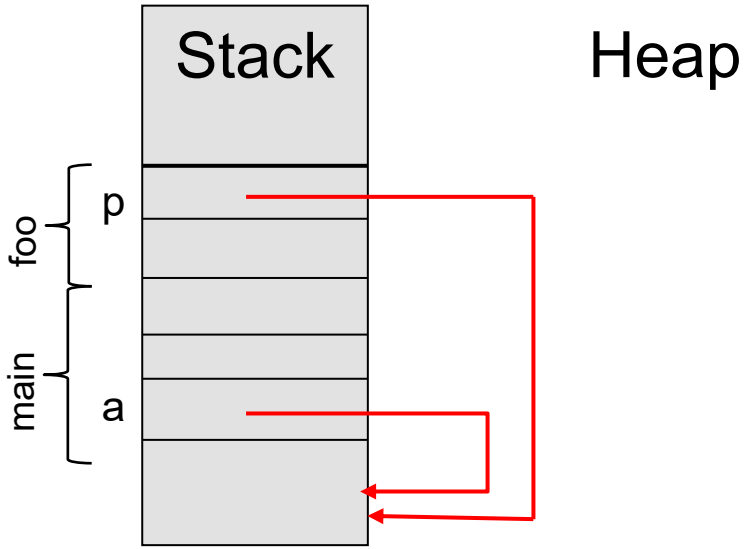
```
void foo(int *p){
  p=malloc(sizeof(int)*N);
}

int main(){
  int *a;
  foo(a);
  ...
```



Allocazione dinamica – sottoprogrammi

- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

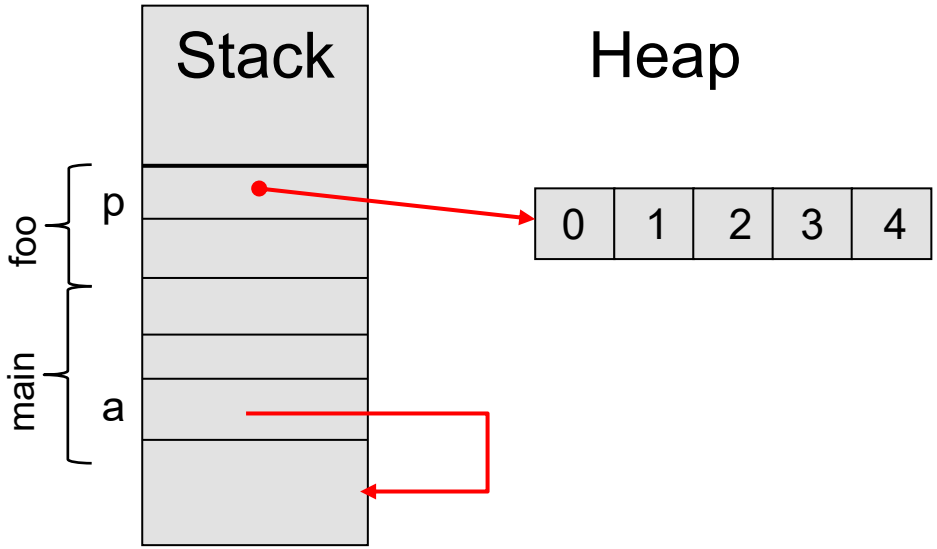
```
void foo(int *p){
  p=malloc(sizeof(int)*N);
}

int main(){
  int *a;
  foo(a);
  ...
```



Allocazione dinamica – sottoprogrammi

- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

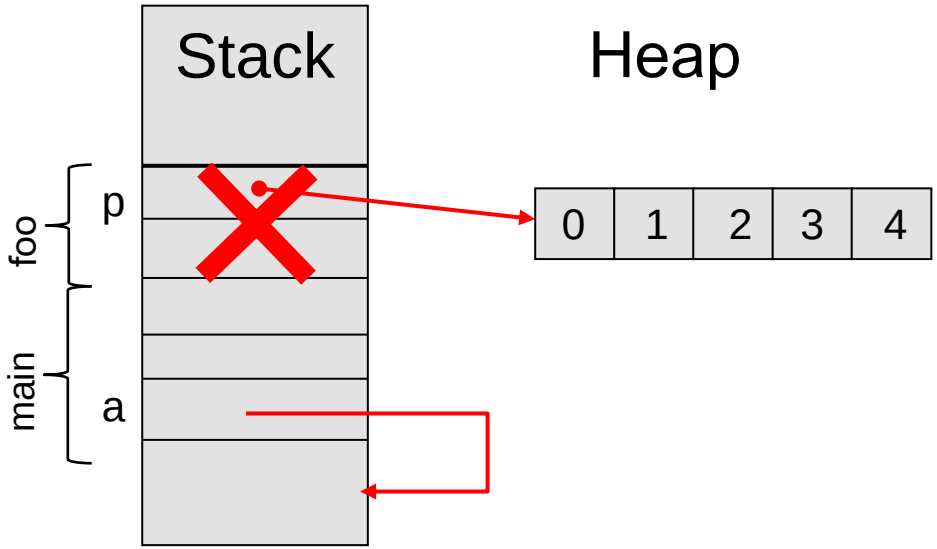
```
void foo(int *p){
  p=malloc(sizeof(int)*N);
}

int main(){
  int *a;
  foo(a);
  ...
```



Allocazione dinamica – sottoprogrammi

- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

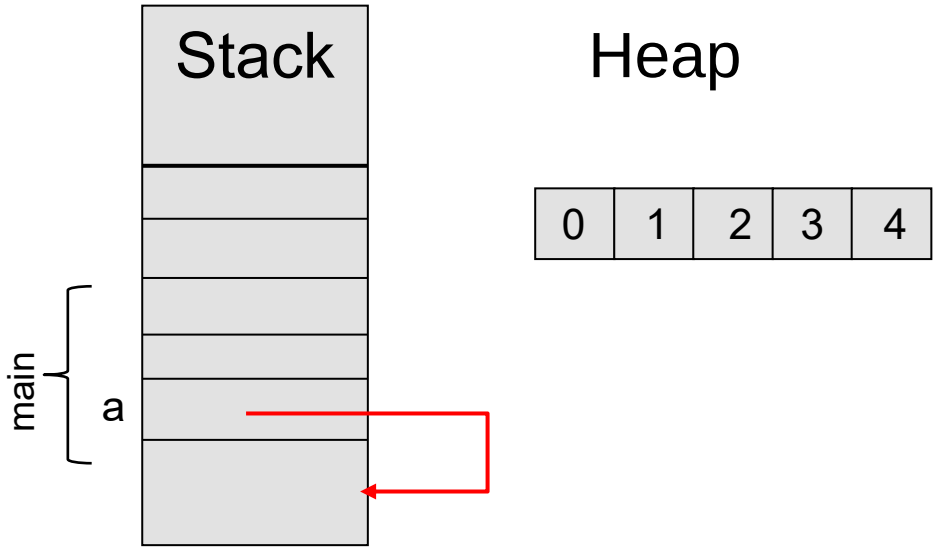
```
void foo(int *p){
  p=malloc(sizeof(int)*N);
}

int main(){
  int *a;
  foo(a);
  ...
```



Allocazione dinamica – sottoprogrammi

- E se invece il main deve passare un puntatore a un sottoprogramma e il sottoprogramma deve modificarlo?

```
void foo(int *p){
  p=malloc(sizeof(int)*N);
}

int main(){
  int *a;
  foo(a);
  ...
```



Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```

Il parametro è un doppio puntatore

Allocazione della memoria ed assegnamento l'indirizzo alla cella puntata dal doppio puntatore (cioè il puntatore nel chiamante)

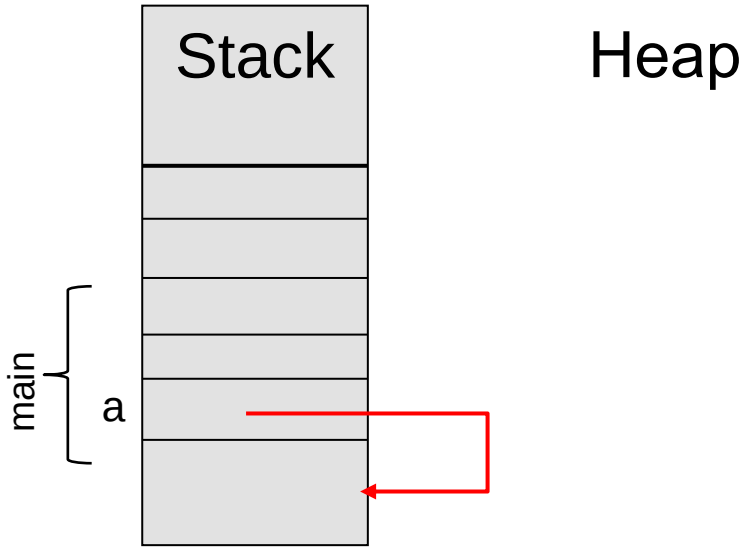
Per accedere alla memoria bisogna dereferenziare due volte (accesso all'elemento con indice 2)

`foo()` è invocata passando l'indirizzo della variabile puntatore in modo tale da permetterne la modifica nel chiamante

Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

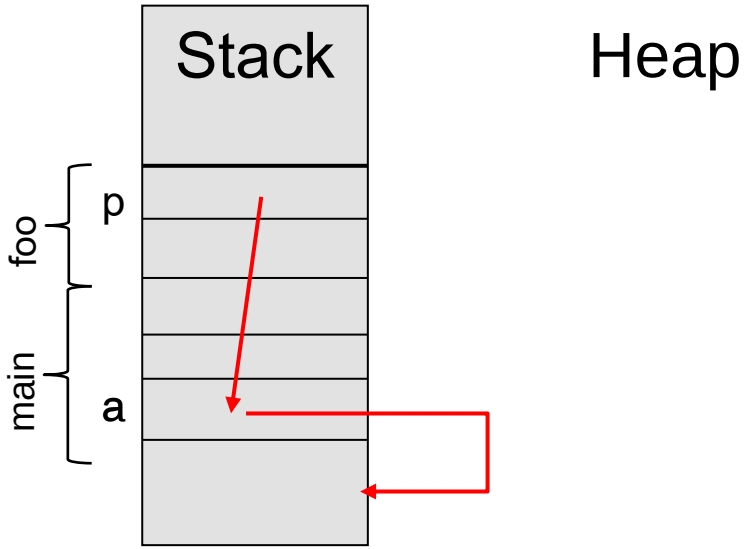
```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```



Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

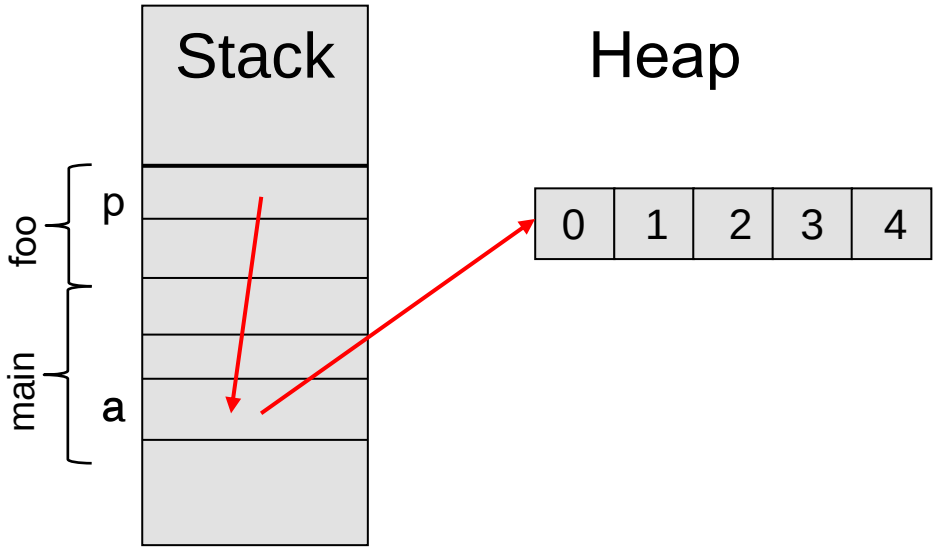
```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```



Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

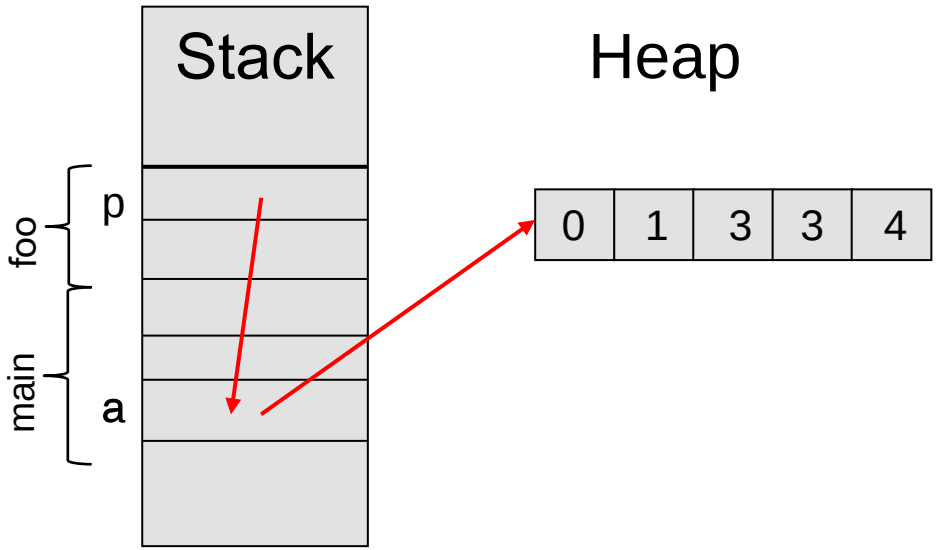
```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```



Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

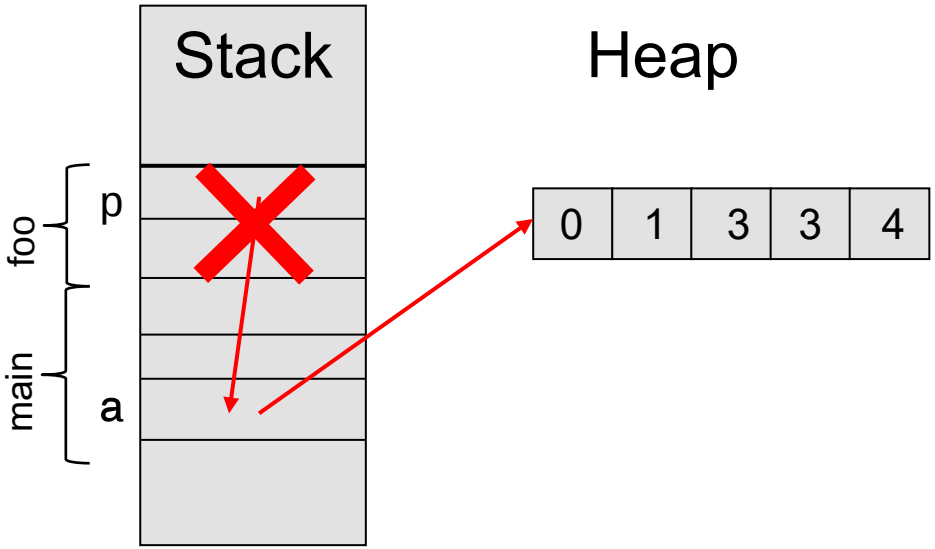
```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```



Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```



Allocazione dinamica

Se si passa ad un sottoprogramma il puntatore come parametro e gli si vuole modificare l'indirizzo contenuto è necessario utilizzare un doppio puntatore

```
void foo(int** p){
   *p=malloc(sizeof(int)*N);
   ...
   *(*p+2)=3;
   ...
}

int main(){
   int *a;
   foo(&a);
   ...
```