

Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Allocazione dinamica della memoria - esercizi introduttivi*

1 La vostra malloc

Considerate il codice contenuto nel file `my-malloc.c`.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 void *my_malloc( size_t n ) {
5     void *p;
6     p = malloc( n );
7     if ( p == NULL ) {
8         printf( ".....\n" );
9         exit( EXIT_FAILURE );
10    }
11
12    return p;
13 }
```

Analizzate il codice e rispondete per iscritto alle seguenti domande.

- A cosa serve la condizione nell'`if` nella riga 7?
- Cosa indica la macro `EXIT_FAILURE`?
- Qual'è l'effetto della funzione `exit` comando nella riga 9?
- Completate la `printf` nella riga 8 con un messaggio opportuno.
- Confrontate il prototipo delle funzione `my-malloc` con quella di `malloc` definita in `stdlib.h`.
- Scrivete un'analogia funzione `my-realloc` con lo stesso prototipo della funzione `realloc`.

2 Tanti elementi

Considerate il codice contenuto nel file `n-elementi.c`. La funzione `read_n` legge un intero `n`, alloca lo spazio per memorizzare `n` interi e li legge da standard input.

```
1 int *read_n( int *num ) {
2     int *a, i;
3     scanf( "%d", num );
4
5     a = my_malloc( *num * sizeof(int) );
6     for ( i = 0; i < *num; i++ ) {
7         scanf( "%d", &a[i] );
8     }
9
10    return a;
}
```

*Ultima modifica 11 novembre 2021

11 }

Analizzate il codice sorgente e rispondete per iscritto alle seguenti domande. Se avete dubbi, potete testarlo, eseguendolo su casi di input significativi e modificandolo.

- Qual è lo scopo del ciclo nelle righe 6-8?
- Riscrivete la riga 7 usando la notazione dei puntatori invece che quella degli array.
- Lo spazio allocato per il puntatore `a` è allocato sullo *stack* o sullo *heap*?
- I valori letti da standard input (scritti in memoria tramite l'invocazione di `scanf` nella riga 7) sono memorizzati nello *stack* oppure nello *heap*? Perché?
- Perché `num` è dichiarato come puntatore a intero?
- Cosa restituisce la funzione?
- Oltre a restituire un valore, la funzione ha degli effetti collaterali (es: produzione di output, modifica dello stato della memoria)?
- Scrivete un `main` per testare la funzione; rappresentate con dei disegni lo stato della memoria subito prima e subito la chiamata della funzione `read_n` nel `main`. Distinguate tre zone: quella che rappresenta lo *heap*, quella che rappresenta il record di attivazione della funzione `read_n` e quella che rappresenta il record di attivazione del `main`.
- Modificate il programma in modo da memorizzare n caratteri invece che n numeri interi. Come cambia la riga 5?

3 Rovescia

Considerate il codice contenuto nel file `rovescia.c`.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #define L 2
5 #define L0 3
6
7 int *read_lin( int *num ) {
8     int *a, size = L0, i = 0, n;
9
10    a = malloc( size * sizeof(int) );
11    while ( 1 ) {
12        scanf( "%d", &n );
13
14        if ( n == 0 )
15            break;
16
17        if ( i >= size ) {
18            size += L;
19            a = realloc( a, size * sizeof(int) );
20        }
21
22        a[i++] = n;
23    }
24    *num = i;
25    return a;
26 }
27
28 int main( ) {
29     int i;
30     int *a = read_lin( &i );
31
32     while ( i-- > 0 )
33         printf( "a[%d] = %d\n", i, a[i] );
34 }
```

Analizzate il codice sorgente e rispondete per iscritto alle seguenti domande. Se avete dubbi, potete testarlo, eseguendolo su casi di input significativi e modificandolo.

- Senza eseguirlo al computer, tracciate l'esecuzione del programma quando riceve il seguente input:
1 2 3 4 5 6 7 8 9 0
- Rappresentate con un disegno lo stato della memoria subito prima della lettura del numero 6. In particolare, quant'è grande lo spazio cui punta `a`?
- Sullo stesso input, quante volte viene eseguita la funzione `realloc`?
- A cosa servono le variabili `size` e `i`?
- Cosa restituisce la funzione `read_lin`?
- A cosa serve il parametro `num`?
- Riassumete con una frase cosa fa il programma.
- Modificate il programma in modo che lo spazio re-allocato raddoppi ogni volta invece di crescere linearmente.

4 Lettura di stringhe con allocazione di memoria

Scrivete due funzioni che leggano da standard input una sequenza di caratteri e la memorizzino in una stringa di dimensione opportuna allocata dinamicamente (scegliete la strategia che preferite, ad esempio una di quelle proposte nell'esercizio precedente):

1. `char *read_line(char c)` deve leggere una riga terminata dal carattere `c`;
2. `char *read_word(void)` deve leggere una parola di caratteri alfanumerici (la lettura deve interrompersi al primo carattere non alfanumerico; se il primo carattere letto non è alfanumerico, la stringa restituita sarà la stringa vuota).

Entrambe le funzioni devono restituire l'indirizzo del primo carattere della stringa memorizzata o il puntatore `NULL` in caso di errore.

5 Matrice

Considerate il codice contenuto nel file `matrice.c`. La funzione `creaMatrice` alloca lo spazio per una matrice di interi.

```
1 char **creaMatrice( int n ){
2     char **m;
3     int r, c;
4
5     m = malloc( n * sizeof( char * ) );
6     for ( r = 0; r < n; r++ ) {
7         *(m+r) = malloc( n * sizeof( char ) );
8     }
9
10    for ( r = 0; r < n; r++ )
11        for ( c = 0; c < n; c++ )
12            m[r][c] = '.';
13    return m;
14 }
```

Analizzate il codice sorgente e rispondete per iscritto alle seguenti domande. Se avete dubbi, potete testarlo, eseguendolo su casi di input significativi e modificandolo.

- A cosa serve l'assegnamento nella riga 5?
- Riscrivete la riga 7 usando la notazione degli array invece della notazione dei puntatori.
- A cosa serve l'assegnamento della riga 7?
- Riscrivete la riga 12 con la notazione dei puntatori anziché quella degli array.
- Rappresentate con un disegno lo stato della memoria prima del `return`.

- Cosa restituisce la funzione?

6 Rettangoli

Completate l'esercizio 8 della scheda `L02-datiAggregati.pdf` con una funzione che crei un nuovo rettangolo allocando lo spazio opportuno, ne assegni i membri con dati inseriti dall'utente, e ne restituisca l'indirizzo; la memoria necessaria va allocata dinamicamente usando la funzione `malloc`.