

Sistemi Operativi - Tutoraggi

Laurea in Ingegneria Informatica

Università Tor Vergata

Tutor: Romolo Marotta

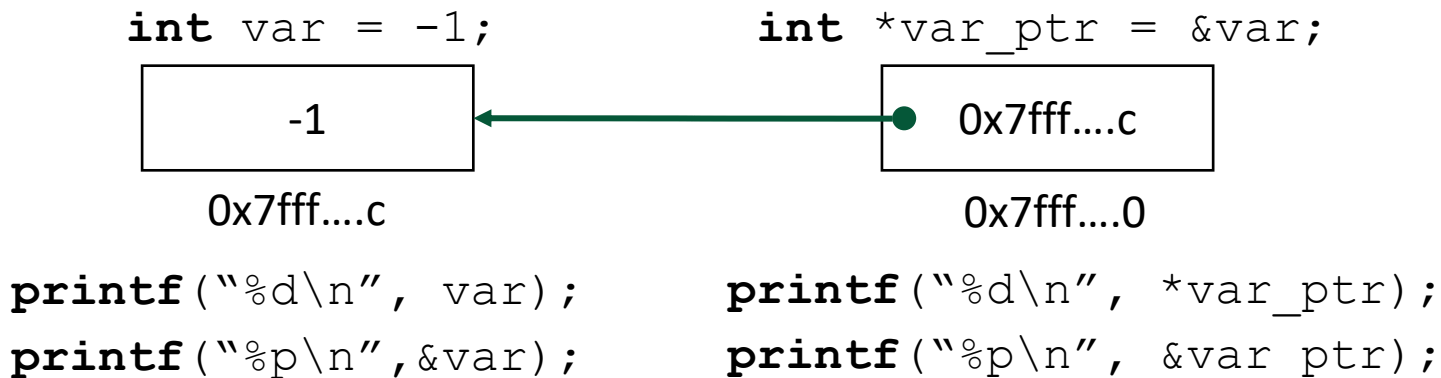
Docente del corso: Francesco Quaglia

Introduzione

1. Variabili puntatore in C
2. printf, scanf
3. gcc
4. Layout programma

I puntatori in C

- Il linguaggio di programmazione C permette al programmatore di accedere esplicitamente allo spazio di indirizzamento di un processo
- Alcuni operatori:
 - * dato un puntatore accede al contenuto della variabile referenziata
 - & data una variabile ne ottiene il puntatore



printf

- `int printf (const char * format, ...);`
- stampa su standard output la stringa *format*
- può prendere argomenti aggizionali che saranno stampati secondo la formattazione specificata dalla stringa *format*
- alcuni specificatori:
 - `%c` : un carattere
 - `%s` : sequenza di caratteri terminanti con `'\0'` (Stringa)
 - `%p` : un puntatore
 - `%d` : signed int decimale
 - `%u` : unsigned int decimale
 - `%x` : esadecimale unsigned
 - `%o` : ottale unsigned

```
printf("La variabile 'var' ha indirizzo %p e valore %d\n", &var, var);
```

The diagram illustrates the argument passing in the `printf` function call. It shows the format string `"La variabile 'var' ha indirizzo %p e valore %d\n"` followed by two arguments: `&var` and `var`. An orange box highlights the `%p` specifier, and a blue box highlights the `%d` specifier. An orange arrow points from the `&var` argument to the `%p` specifier. A blue arrow points from the `var` argument to the `%d` specifier. A blue line connects the two boxes above the comma, indicating the sequence of arguments.

Esempio 1

scanf

- `int scanf(const char * format, ...);`
- scansiona l'input in accordo alla stringa *format*
- la stringa *format* nel può prendere argomenti addizionali che saranno stampati secondo la formattazione specificata dalla stringa *format*

The diagram illustrates the mapping between format specifiers in a `scanf` format string and the corresponding variables. In the code `scanf ("%s:%s\n", h, m);`, the format string is enclosed in an orange box, and the variables `h` and `m` are each enclosed in a blue box. An orange arrow points from the `%s` specifier in the format string to the `h` variable box. A blue arrow points from the second `%s` specifier in the format string to the `m` variable box.

```
scanf ("%s:%s\n", h, m);
```

Esempio 2

scanf

- `int scanf(const char * format, ...);`
- scansiona l'input in accordo alla stringa *format*
- la stringa *format* nel può prendere argomenti addizionali che saranno stampati secondo la formattazione specificata dalla stringa *format*

The diagram illustrates the mapping between format specifiers in a `scanf` call and the variables they store data into. The code snippet is `scanf ("%s:%s\n", h, m);`. The format string is enclosed in quotes. Within it, `%s` is highlighted in orange and `%s` is highlighted in blue. The variable `h` is enclosed in an orange box, and the variable `m` is enclosed in a blue box. An orange arrow points from the orange `%s` to the orange box `h`. A blue arrow points from the blue `%s` to the blue box `m`. Additionally, a blue arrow points from the `%s` to the `%s`, indicating that the second `%s` is a continuation of the first one.

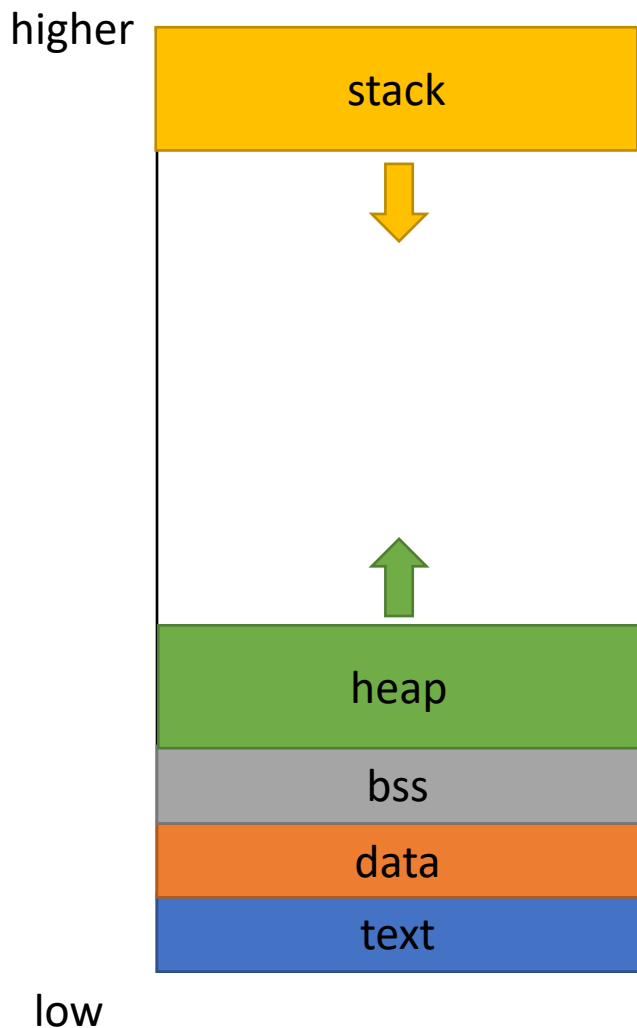
```
scanf ("%s:%s\n", h, m);
```

- il prototipo di uno specificatore di formato è:

`%[*][width][length]specifier`

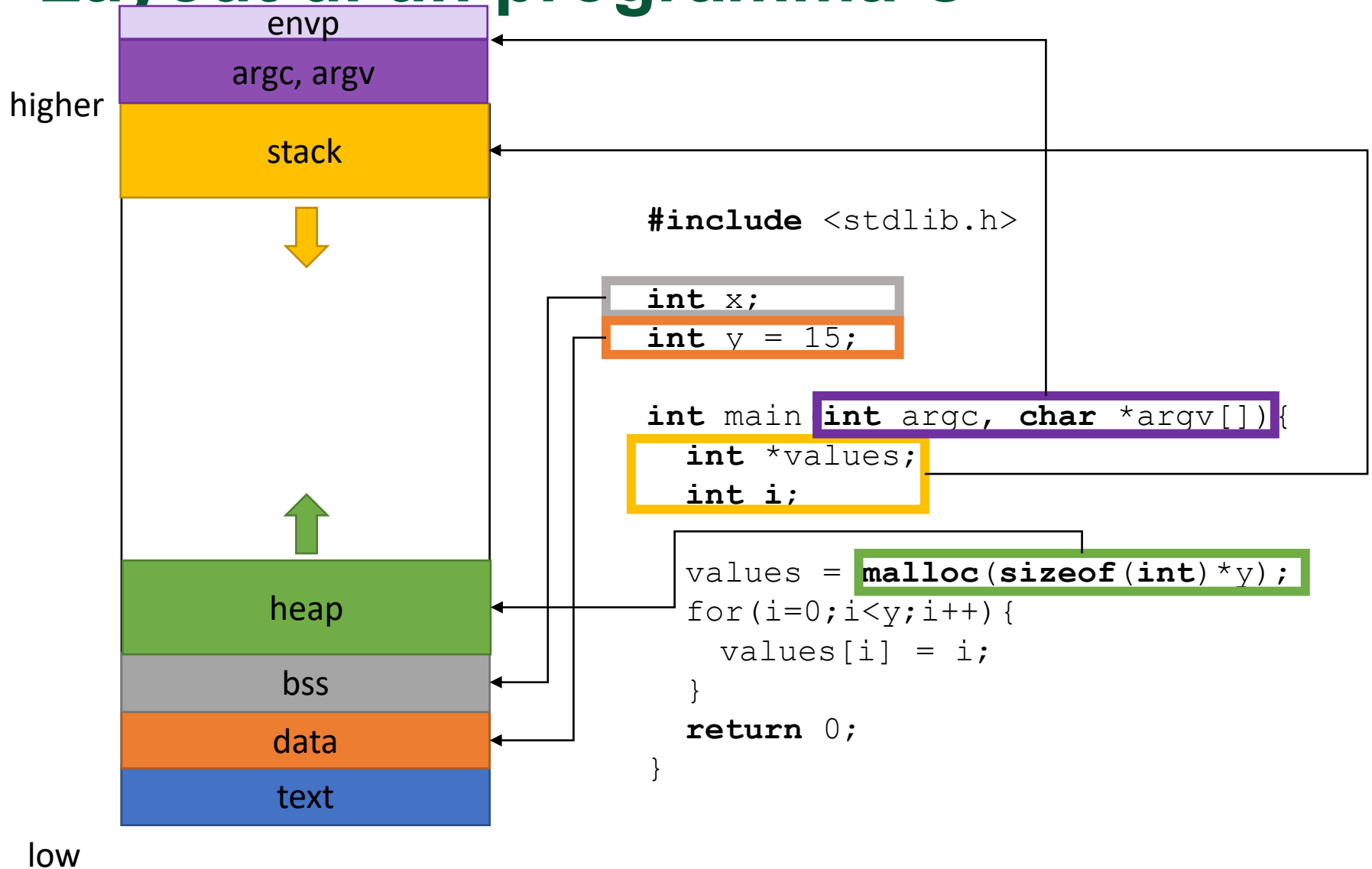
Esempio 3

Layout di un programma C



- **Text**: istruzioni eseguibili
- **Data**: dati inizializzati
- **Block started by symbol (BSS)**: dati non inizializzati o inizializzati al valore zero
- **Heap**: sezione di dati allocati dinamicamente
- **Stack**: per chiamate a procedura, passaggio parametri, indirizzo di ritorno, variabili locali

Layout di un programma C



Esercizio 1

- Scrivere un programma che prende una stringa da tastiera e la inserisce all'interno di un buffer allocato dinamicamente nella heap da parte della funzione `scanf()`.
- Copiare poi tale stringa all'interno di un secondo buffer allocato sullo stack della taglia necessaria a contenerla.
- Liberare quindi il buffer allocato nella heap utilizzando la funzione `free()`.
- Stampare sullo schermo la stringa copiata nel buffer allocato sullo stack.

Esercizio 2

- Scrivere un programma che prende una stringa passata come primo argomento (i.e. `char *argv[]`) al programma stesso quando questo viene eseguito.
- Copiare tale stringa all'interno di un buffer di dimensione fissa facendo attenzione a non superare il limite imposto dalla taglia, e stamparla quindi sullo schermo.
- Rigidare la stringa (primo carattere in ultima posizione, secondo carattere in penultima posizione, ecc.) senza fare utilizzo di un ulteriore buffer per poi stampare anche questa stringa sullo schermo.