



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Chiamate a funzione e memory layout

Stefano Ghidoni



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE



Agenda

- Meccanismo di chiamata a funzione
- Layout di memoria di un processo
- Gestione delle variabili in memoria



Chiamata a funzione

- Una chiamata a funzione causa la creazione di una struttura dati chiamata RA – Record di Attivazione (Function Activation Record) che contiene:
 - una copia dei parametri
 - variabili locali
- I RA sono impilati in una zona di memoria chiamata stack
 - Stack (pila): struttura LIFO (è l'altro significato informatico rispetto a quello che fa riferimento alla zona di memoria)
- I RA hanno dimensione che dipende dal numero (e dal tipo) di variabili locali
 - Il tempo necessario per inserire il RA nello stack è costante



```
int f(int f1, int f2)
{
    int var_f1;
    int var_f2;
    // ...
}

int g(int g1)
{
    double var_g1;
    // ...
}
```



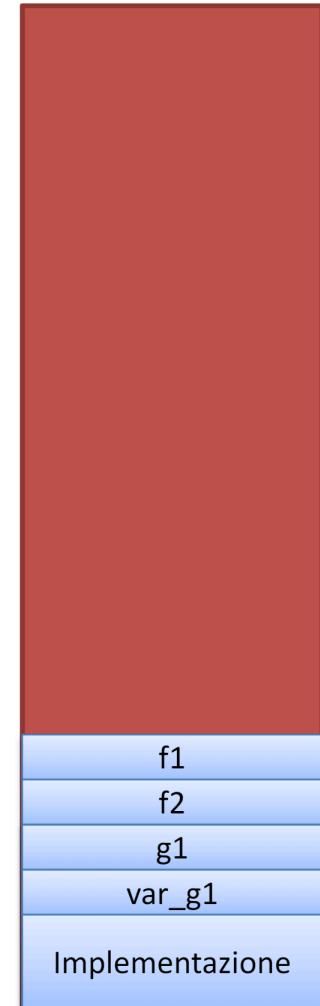
- Implementazione: informazioni che servono per ritornare al chiamante e fornire un risultato (return)
- Parametri e variabili locali sono equivalenti in questo schema
- Ogni chiamata a f o g ha il proprio RA

serve per gestire il ritorno alla funzione chiamante
etc. (vedi architettura degli elaboratori)



Meccanismo di chiamata a funzione

- Stack
- Chiamate in sequenza
 - Chiamata a f
 - f esce
 - Chiamata a g
 - g esce





Meccanismo di chiamata a funzione

- Stack
- f chiama g e viceversa
 - Chiamata a f
 - Chiamata a g
 - Chiamata a f (2)
 - » Chiamata a g (2)
 - » g (2) esce
 - f (2) esce
 - g esce
 - f esce

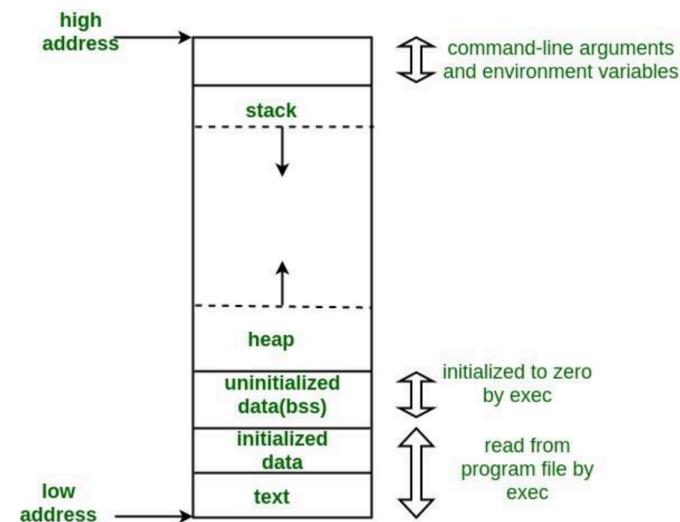


c'è anche il record di attivazione del main



Layout di memoria

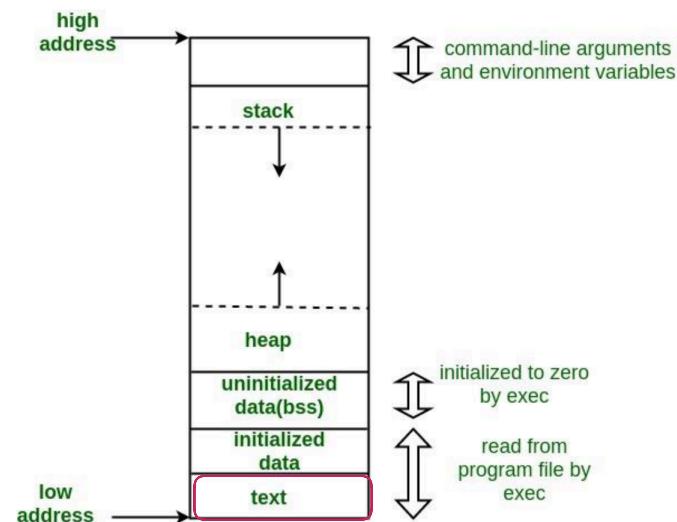
- Text segment
- Initialized data segment
- Uninitialized data segment
- Stack (per le variabili locali)
- Heap /free store





Layout di memoria

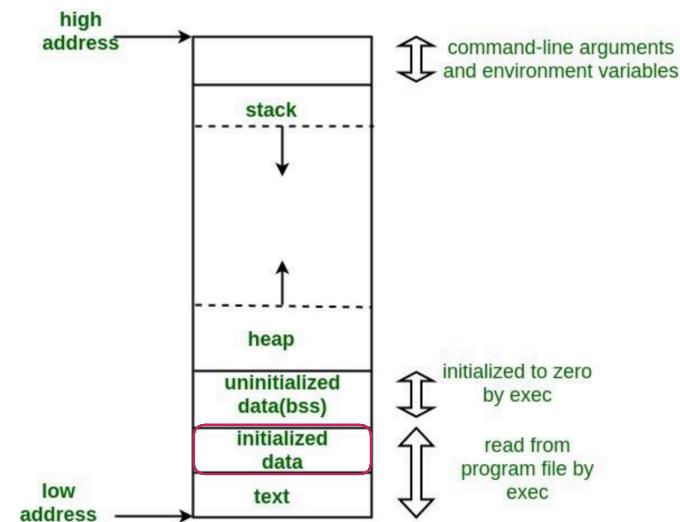
- Text segment
 - AKA **code** segment / text
 - Copiato in memoria dal file oggetto
 - Contiene le istruzioni eseguibili
 - Spesso read-only





Layout di memoria

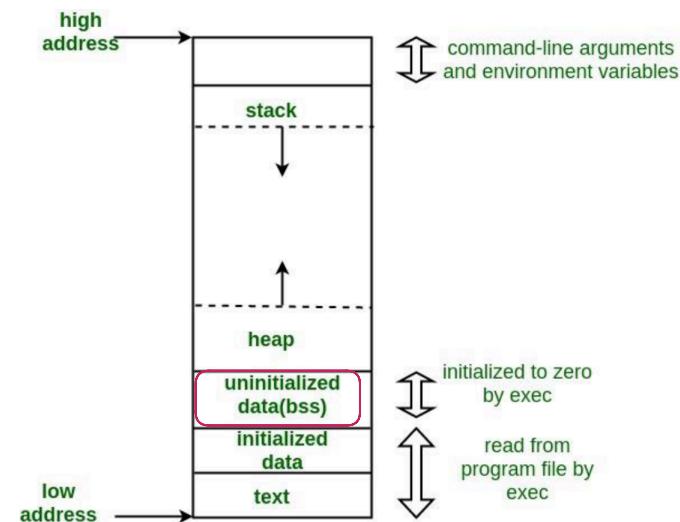
- Initialized data segment
 - AKA data segment
 - Contiene variabili globali e variabili statiche
 - Read/write
 - Può contenere una parte read-only





Layout di memoria

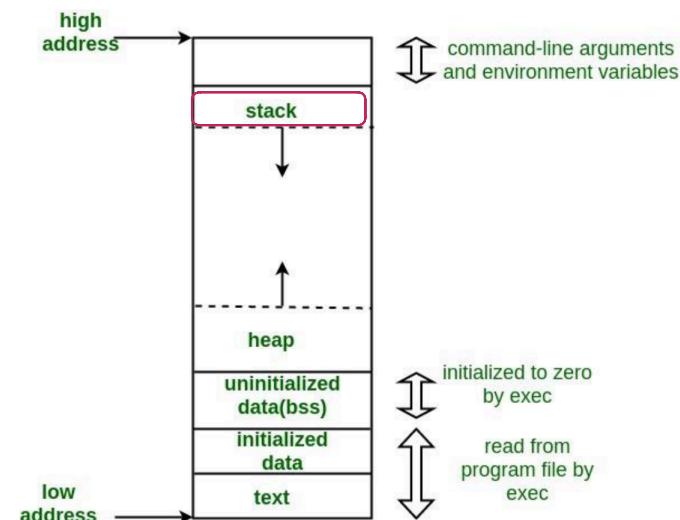
- Uninitialized data segment
 - AKA BSS segment (Block Started by Symbol, ragioni storiche)
 - Inizializzato a 0 dal SisOp
 - Contiene variabili globali e statiche non inizializzate esplicitamente





Layout di memoria

- Stack
 - Contiene i RA delle funzioni
 - Quindi le variabili locali e i parametri





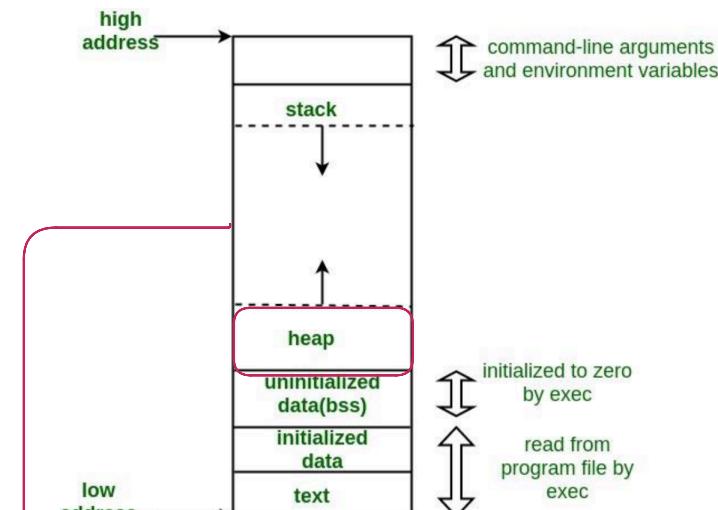
Layout di memoria

- **Heap /free store**

- Formalmente è spesso visto come uno spazio che cresce in verso opposto allo stack
- La sua dimensione può essere modificata con le funzioni di sistema brk e sbrk
- Può essere gestito usando blocchi di memoria non contigui (funzione di sistema mmap)
- Gestito tramite **new** e **delete**
 - Allocazione dinamica della memoria!

Programma: ricetta

Processo: cuoco che cucina secondo la segue la ricetta.



Lo spazio dello stack e dell'heap non è condiviso, questo "rettangolo" può crescere.

Lo stack ha una memoria fissa massima, mentre l'heap può crescere finché non occupa tutta la memoria del sistema.



Valutazione delle variabili

```
string program_name = "silly";  
vector<string> v;
```

Variabili globali

```
void f()  
{  
    string s;          // s locale a f  
    while (cin >> s && s != "quit") {  
        string stripped; // locale nel loop  
        string not_letters;  
        for (int i = 0; i < s.size(); ++i)  
            if (isalpha(s[i]))  
                Variabili locali  stripped += s[i];  
            else  
                not_letters += s[i];  
        v.push_back(stripped);  
        // ...  
    }  
    // ...  
}
```

Molti tipi di variabili
Che cosa le differenzia
(a parte il tipo)?



Durata / classe di memorizzazione

- Una importante differenza riguarda la classe di memorizzazione (o durata)
 - **Durata:** per quanto tempo la variabile è in memoria?
- La durata è diversa dallo scope
 - **Scope:** in quale parte del codice la variabile è accessibile?



- Variabili statiche
 - Inizializzate solo la prima volta
 - Mantengono il valore anche al di fuori del loro scope
 - Possono essere globali o locali (se dichiarate dentro a una funzione)
 - Variabile globale: sempre statica
 - Variabile locale: statica se dichiarata static

Si trova in memoria da prima dell'esecuzione della prima istruzione del main, rimane fino alla fine dell'esecuzione.

static int i = 100 → questo pezzo di codice dentro una funzione significa che la variabile viene inizializzata a 100 solo la prima volta che viene inizializzata, quando cambia le variabili non sono più accessibili.



Durata / classe di memorizzazione

- Variabili globali (program_name, v)
 - Iniziate prima della prima istruzione del main
 - Esistono fino al termine del programma
 - Sono **statiche**

```
string program_name = "silly";
vector<string> v;

// ...
```



Durata / classe di memorizzazione

- Variabili locali automatiche
 - A ogni chiamata di f(), si costruisce s
 - s inizializzata alla stringa vuota
 - s esiste fino all'uscita da f()
 - A ogni iterazione del loop, si costruiscono stripped e not_letters
 - Esistono fino alla fine del loop
 - Distrutte in ordine inverso alla creazione

```
// ...  
  
void f()  
{  
    string s;           // s locale a f  
    while (cin >> s && s != "quit") {  
        string stripped;      // locale  
        string not_letters;  
    // ...  
}
```



Variabili statiche vs automatiche

```
string program_name = "silly";           // ?
vector<string> v;                      // ?

void f()
{
    int local_auto = 0;                  // ?
    static int local_static = 0;         // ? Mantiene il suo valore

    cout << "Auto: " << local_auto++ << ", static: "
        << local_static++ << '\n';
}
```

A ogni interazione local_auto riparte da 0, mentre local_static no.
static int local_static = 0 → viene eseguito una sola volta.



Valutazione delle variabili

- I compilatori fanno sì che il codice si comporti come se le azioni descritte prima fossero eseguite
 - Varie ottimizzazioni sono possibili
 - I compilatori spesso modificano l'implementazione per ottimizzare
 - Ma il comportamento è immutato



Inizializzazione globale

- Le variabili globali sono inizializzate prima dell'esecuzione della prima istruzione del main
 - Se le variabili di diverse **translation unit** hanno una dipendenza, non sappiamo in che ordine sono effettuate le inizializzazioni
 - Variabili globali hanno la memoria scritta a 0 prima delle inizializzazioni



blocchi di codice post-preprocessore



Inizializzazione globale

```
// file f1.cpp - variabili globali
int x1 = 1;
int y1 = x1 + 2;
```

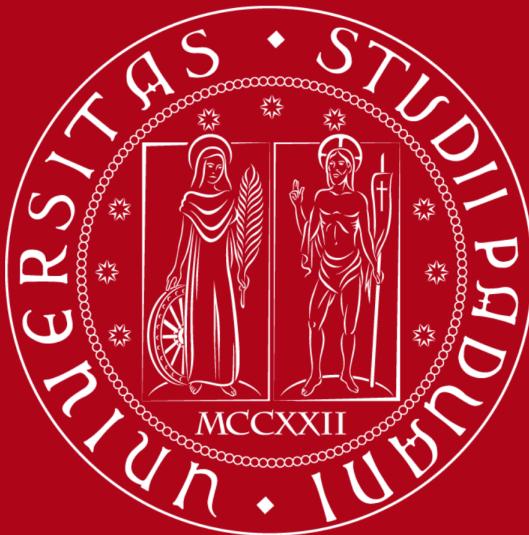
```
// file f2.cpp - variabili globali
extern int y1;
int y2 = y1 + 2;
```

// n.b.: codice da evitare!!



Recap

- Record di attivazione, ruolo nella chiamata a funzione
- Layout della memoria
 - Varie regioni, ciascuna con la sua caratteristica
- Variabili
 - Locali vs globali
 - Statiche vs automatiche
- Inizializzazione globale



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Chiamate a funzione e memory layout

Stefano Ghidoni



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE