DIETI UNINA UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea in Ingegneria Informatica

Corso di Ingegneria del Software

Richiami di Programmazione

Classi di memorizzazione in C++

Riferimenti

- C. Savy: Da C++ a UML - Guida alla progettazione
  - Capitolo 3 § 1;
  - Capitolo 12 § § 2-10.

Dichiarazioni e definizioni – 1/5

- È importante distinguere tra dichiarazioni e definizioni delle "entità" di un programma (costanti, variabili, funzioni, tipi, etc.).

- Una **dichiarazione** è una frase che introduce un nome in un programma e ne enuncia delle proprietà
  - Per costanti e variabili:
    - Enuncia il tipo, la classe di memorizzazione;
  - Per nomi di tipo:
    - Introduce il nome e la categoria (es.:, struct, class, etc.);
  - Per le funzioni:
    - Prototipo.

Dichiarazioni e definizioni – 2/5

- Una **definizione** definisce completamente un'entità:
  - Per le variabili:
    - Dichiara il tipo e definisce allocazione in memoria;
  - Per nomi di tipo:
    - Definisce tutti i dettagli del tipo;
  - Per le funzioni:
    - Definisce il corpo (codice) della funzione.

Dichiarazioni e definizioni – 3/5

- Una definizione è anche una dichiarazione.
- Variabili, funzioni, etc. vanno dichiarate prima di essere utilizzate (nei linguaggi tipizzati).
- Per ogni entità possono esistere più dichiarazioni.
- Le definizioni invece non possono essere ripetute.
- Le dichiarazioni devono essere consistenti tra loro e con le definizioni.

Dichiarazioni e definizioni – 4/5

Esempi di *dichiarazioni*:

```
extern int a; // dichiara(ma non defin.) la variabile a,
              // definita in altro file

struct Persona; // dichiara un nome di tipo per un record

void f(int); // dichiara(ma non definisce) la
             // funzione f (prototipazione)
```

Dichiarazioni e definizioni – 5/5

Esempi di *definizioni*:

```
const int b = 5; // defin. di costante con nome

int a; // defin. di variabile

struct Persona { // definisce il tipo Persona
  char Nome[20];
   … } ;

void f(int x) { // definizione di funzione
   …            // corpo della funzione
}
```

Classi di memorizzazione delle variabili in C++

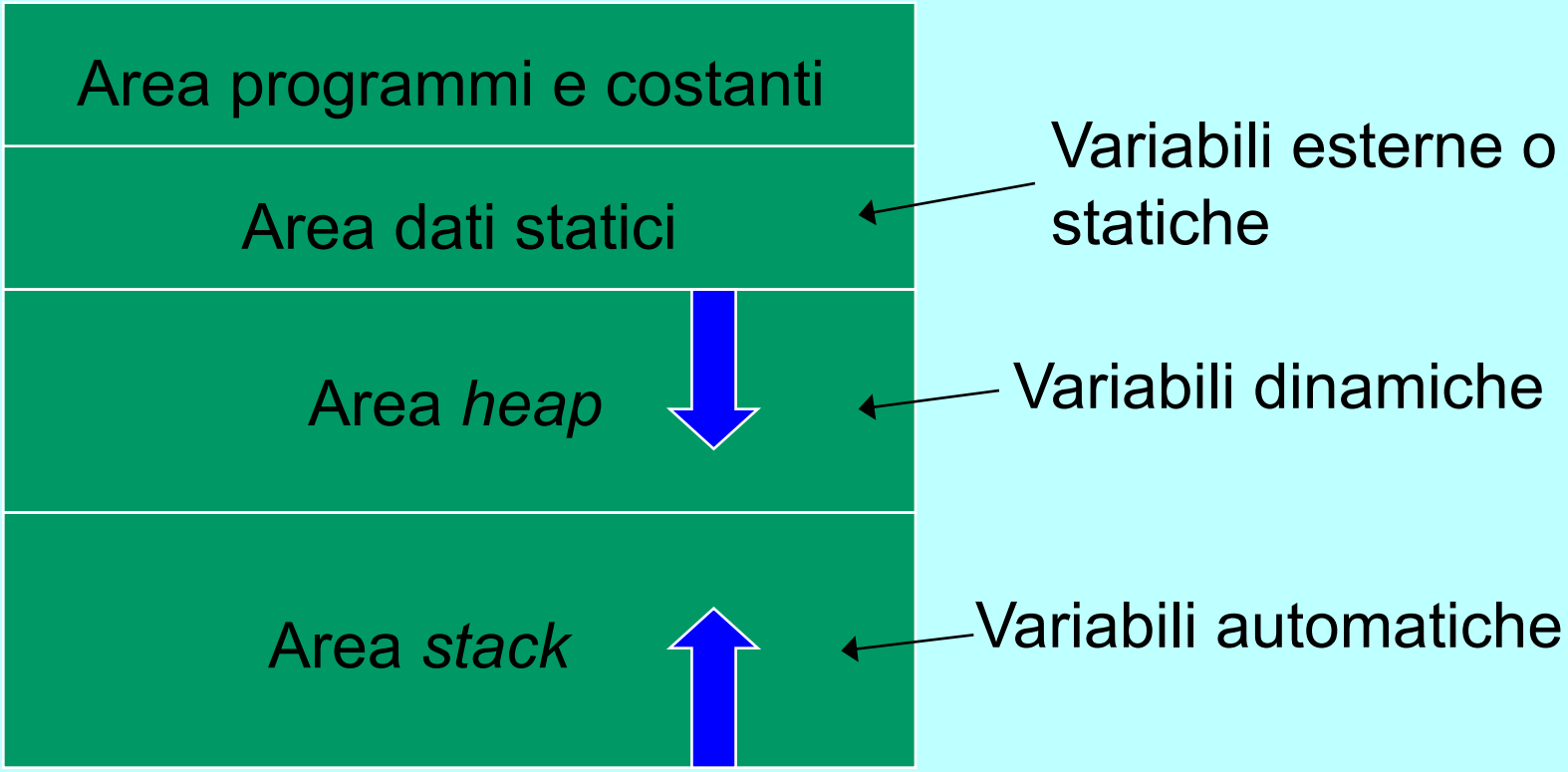
- Per le variabili in C++ la definizione specifica la classe di memorizzazione, cioè:
  - la visibilità spaziale;
  - la durata temporale (ciclo di vita);
  - l'allocazione in memoria durante l'esecuzione.

- Classi di memorizzazione del C++ (*storage classes*):
  - variabili automatiche;
  - variabili esterne;
  - variabili automatiche statiche;
  - variabili esterne statiche;
  - variabili *register;*
  - variabili dinamiche.

Modello di gestione della memoria per l'esecuzione di un processo – 1/2

Un processo è un programma in esecuzione.

Area di memoria allocata per un processo



Modello di gestione della memoria per l'esecuzione di un processo – 2/2

Area programmi e costanti
destinata a contenere le istruzioni (in linguaggio macchina) e le costanti del programma.

Area dati statici
destinata a contenere variabili allocate staticamente e quelle esterne.

Area *heap* (a mucchio)
destinata a contenere variabili dinamiche esplicite, di dimensioni non prevedibili a tempo di compilazione.

Area *stack*
Destinata a contenere le variabili automatiche e quelle definite all'interno delle funzioni.

Variabili automatiche

- Sono le variabili locali ad un blocco di istruzioni.
- Sono definite in un blocco di istruzioni e sono allocate e deallocate con esso.
- La visibilità lessicale è locale al blocco.
- Nell'immagine in memoria del processo (cioè del programma in esecuzione), sono allocate nell'area *stack*.

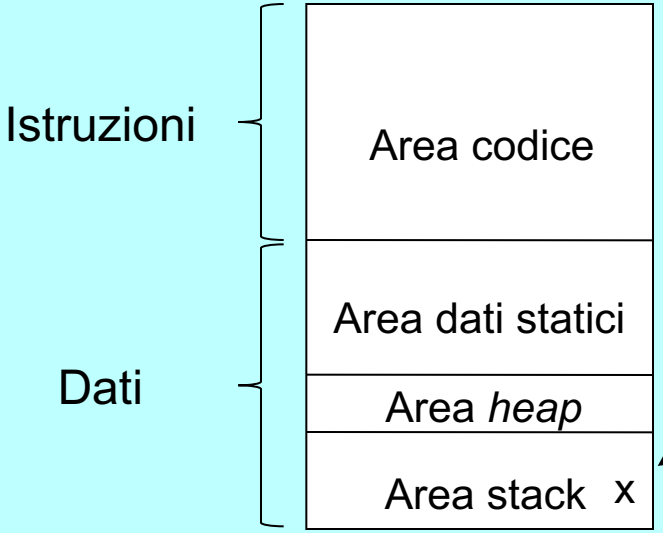


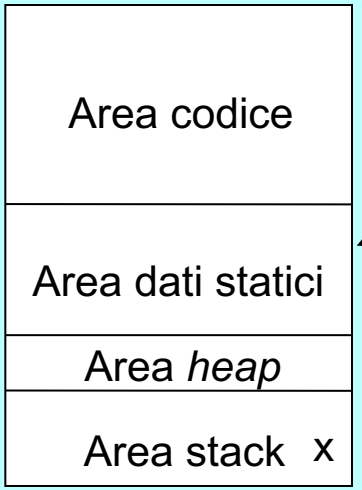
Immagine di un processo in memoria

Esempio:

```
{
    int x;   // var. autom.
    …
}
```

Variabili esterne

- Le variabili esterne sono definite all'esterno di ogni blocco.
- La visibilità lessicale si estende a tutto il programma, ma per renderle visibili anche da procedure contenute in file diversi, devono essere ivi dichiarate con la parola chiave *extern.*
- La loro estensione temporale si estende a tutto il programma (sono allocate all'inizio del programma per essere poi deallocate alla fine dello stesso).
- In memoria vengono allocate nell'area dati statici.



Esempio:

```
int x;   // variabile esterna

void main () {
    …
}
```

Variabili automatiche statiche

- Sono variabili automatiche prefissate con la parola chiave *static.*
- La visibilità lessicale è locale al blocco in cui sono definite.
- La loro estensione temporale si estende a tutto il programma (sono allocate all'inizio del programma per essere poi deallocate alla fine dello stesso).
- In memoria vengono allocate nell'area dati statici.

Esempio:

```
void f() {
    static int x;   // var. autom.
                    //statica
    …
}
```

Variabili esterne statiche

- Sono variabili esterne prefissate con la parola chiave *static.*
- La visibilità lessicale è locale al file in cui sono definite.
- La loro estensione temporale si estende a tutto il programma (sono allocate all'inizio del programma per essere poi deallocate alla fine dello stesso).
- In memoria vengono allocate nell'area dati statici.

Esempio:

```
static int x;   // var. esterna
                // statica

void main () {
    …
}
```

Variabili *register*

- Sono variabili automatiche dichiarate con la parola chiave *register.*
- La visibilità lessicale è locale al blocco in cui sono definite.
- Sono allocate e deallocate con il blocco di istruzioni in cui sono contenute.
- Se possibile vengono allocate in un registro del processore, altrimenti nell'area *stack*.
- Può convenire dichiarare *register* una variabile automatica usata di frequente, in modo che l'allocazione in un registro di macchina ne aumenti la velocità di accesso.

Variabili dinamiche

- Sono definite durante l'esecuzione del programma mediante *puntatori.*
- Sono accessibili dovunque esiste un riferimento ad esse.
- L'allocazione e la deallocazione sono controllate dal programmatore.
- In memoria vengono allocate nell'area *heap*.

Riepilogo – 1/2

Classe di memorizzazione	Definizione	Visibilità	Durata	Allocazione
Automatiche	Definite in un blocco	Blocco	Blocco	Area *stack*
Esterne	Definite all'esterno di ogni blocco	Tutto il programma	Tutto il programma	Area dati statici
Automatiche statiche	Definite in un blocco + *static*	Blocco	Tutto il processo	Area dati statici
Esterne statiche	Definite all'esterno di ogni blocco + *static*	All'interno del file in cui sono definite	Tutto il processo	Area dati statici
Register	Definite in un blocco + *register*	Blocco	Blocco	Se possibile, in un registro del processore
Dinamiche	Tramite puntatori e operatori new/delete	Finché esiste un riferimento valido	Definita dal programmatore	Area *heap*

Riepilogo – 2/2