Università degli Studi di Cassino e del Lazio Meridionale

Corso di Fondamenti di Informatica

Visibilità e tempo di vita delle variabili

Anno Accademico 2016/2017

Francesco Tortorella

Il concetto di visibilità

- Un programma C++ può assumere una struttura complessa grazie all'uso dei sottoprogrammi (programma costituito da più sottoprogrammi su più file sorgente).
- Questo rende necessario definire delle regole per l'uso degli oggetti (variabili, costanti, funzioni) definite nel programma.
- In particolare, tali regole precisano in quali parti del programma è possibile usare un certo identificatore (**visibilità**).

Visibilità delle variabili. Ambiente

- L'insieme delle variabili definite in un sottoprogramma (sia funzione che procedura) può dividersi in due insiemi:
  - Parametri formali: utilizzati per gestire il flusso di dati con il chiamante
  - Altre variabili utilizzate per implementare l'algoritmo nel sottoprogramma (es. indici, variabili di appoggio, ecc.
- L'insieme di queste variabili viene definito *ambiente* del sottoprogramma.
- Analogamente, l'insieme delle variabili definite nel chiamante costituisce l'ambiente del chiamante.

Ambienti e visibilità

- Quale relazione esiste tra ambiente del chiamante e ambiente del sottoprogramma ?
- In altre parole, il sottoprogramma ha la **visibilità** (cioè, può fare uso) delle variabili del chiamante e viceversa ?

Ambienti e visibilità

- Sono **due ambienti distinti** per cui:
  - Le variabili del chiamante non sono visibili dal sottoprogramma e viceversa.
  - Nei due ambienti possono quindi esistere variabili con lo stesso nome, ma sono due variabili distinte e separate.
  - L'unico canale per scambiarsi dati è quindi fornito dallo scambio di parametri.

Visibilità delle variabili

- Le variabili sono **locali** alle funzioni: sono cioè utilizzabili solo all'interno della funzione in cui sono definite.
- Più precisamente, la visibilità si riferisce al **blocco** in cui sono definite.

Visibilità delle variabili

```
#include<iostream>
using namespace std;

int doppia(int x);

int main() {
     int a,b,c;

     cout << "a: "; cin >> a;
     b=doppia(a);
     c=doppia(b);
     cout<<"a: "<<a<<endl;
     cout<<"b: "<<b<<endl;
     cout<<"c: "<<c<<endl;
     return(EXIT_SUCCESS);
}

int doppia(int x) {
     int a;

     a=x;
     {
        int c;

        c=a;
        a=2*c;
     }
     return(a);
}
```

Visibili `a`, `b` e `c` della funzione `main`

Visibile `a` della funzione `doppia`

Visibili `a` e `c` della funzione `doppia`

Visibilità delle variabili

- All'interno di ogni blocco sono visibili:
  - le variabili definite al suo interno
  - le variabili definite nel blocco che eventualmente lo contiene
- La definizione di una variabile in un blocco annulla la visibilità di eventuali variabili con lo stesso nome, ma definite in blocchi esterni.

Variabili definite nel file

- E’ possibile definire variabili anche al di fuori di ogni funzione, nel file. Tali variabili saranno visibili in tutto il file, dal punto in cui sono definite in poi.
- La visibilità può estendersi anche ad altri file che, eventualmente, costituiscono il programma.

Visibilità delle variabili

```
#include<iostream>
using namespace std;

int doppia(int x);
int tripla(int x);
int z;

int main() {
     int a,b,c;

     cout << "a: "; cin >> a;
     b=doppia(a);
     c=tripla(a);
     cout<<"a: "<<a<<endl; cout<<"b: "<<b<<endl;
     cout<<"c: "<<c<<endl;
     return(EXIT_SUCCESS);
}

int doppia(int x) {
     int a;
     a=2*x;
     return(a);
}

int tripla(int x) {
     int z;
     z=3*x;
     return(z);
}
```

Visibilità di `z`

Visibilità di `z` della funzione `tripla`

Variabili visibili ad altri file

- Assumiamo che il codice di un programma sia distribuito su 3 file: fil1.cpp, fil2.cpp e fil3.cpp.
- Una variabile definita in uno dei file, esternamente alle funzioni, può essere visibile anche agli altri file (e alle funzioni in essi contenute).
- È però necessario introdurre nel file che la userà una **dichiarazione** della variabile tramite la parola chiave **`extern`**.
  Esempio:

```
int conta;
```

```
extern int conta;
```

Variabili visibili ad altri file

```
#include <iostream>
#include "calcol.h"
using namespace std;

int conta=0;

int main() {
    const float prec=1e-8;
    float x,y;
    float d;

    cout<<"\nCoordinate\n";
    cout<<"x: "; cin >>x;
    cout<<"y: "; cin >>y;

    d=radq(x*x+y*y,prec);

    cout<<"\nDistanza: "<<d<<endl;
    return (EXIT_SUCCESS);
}
```

```
extern int conta;

float assol(float x){
    if(x<0)
        return(-x);
    else
        return(x) ;
}

float radq(float x,float p){
    float yn,yv;
    conta=1;
    yn=1.0;

    do{
        yv=yn;
        yn=.5*(yv+x/yv);
    } while (assol(yn-yv)>=p);

    return(yn);
}
```

Tempo di vita delle variabili

- Un'altra questione riguarda quando una variabile può essere usata, cioè in quale momento della vita del programma alla variabile è allocata memoria.
- Questo si formalizza con il concetto di **classe di memorizzazione** (**storage class**) delle variabili.

Variabili automatiche

- Le variabili definite in un blocco si definiscono variabili automatiche in quanto vengono allocate all'inizio dell'esecuzione del blocco e deallocate al termine.
- La conseguenza principale è che queste variabili non conservano il loro valore tra due esecuzioni successive del blocco in quanto generalmente allocate su registri diversi.

Variabili statiche

- Le variabili definite all'esterno delle funzioni sono allocate all'inizio dell'esecuzione del programma e deallocate al termine. Conservano quindi il loro contenuto per l'intera durata del programma (variabili statiche).
- È possibile rendere statica una variabile automatica tramite lo specificatore **`static`**.

Variabili statiche

```
void contachiama() {
  static int count = 0;
  count++;
  cout << "Sono stata chiamata ";
  cout << count << " volte\n";
}

int main() {
  for(int i=0; i<10; ++i )
    contachiama();
}
```

Specificatori della classe di memorizzazione

`auto`: definisce una variabile automatica (puramente opzionale)

`static`: definisce una variabile statica

`register`: specifica al compilatore che la variabile sia allocata in uno dei registri interni del processore