Funzioni e puntatori a variabili o strutture dati.

Vitoantonio Bevilacqua

vitoantonio.bevilacqua@poliba.it

Parole chiave: Dichiarazione, **chiamata** e **definizione** di una **funzione**, **passaggio per valore** di argomenti a una funzione, le **variabili puntatore** dichiarazione e significato, utilizzo e caratteristiche del passaggio di argomenti a funzioni tramite il valore del loro indirizzo, **puntatori a vettori** dichiarati staticamente e relativo passaggio alle funzioni, **puntatori a matrici** dichiarate staticamente e relativo passaggio alle funzioni, **stringhe** quali vettori di caratteri e relative funzioni elementari della libreria **string.h**, **strlen**, **strcpy**, **strcmp**, **strcat**, **dati strutturati** e **struct e typedef, vettori di elementi di dati strutturati** e passaggio alle funzioni.

Introduzione.

Con questa lezione iniziamo il corso di Laboratorio di Informatica da 6 CFU. Questa dispensa costituisce quindi il primo esempio di materiale didattico che si intende mettere a disposizione degli studenti, seguendo l'ordine e le modalità con cui gli argomenti verranno presentati a lezione. Obiettivo dell'intero corso è mettere in condizione lo studente frequentante, destinatario elettivo di queste dispense, ma anche lo studente non frequentante, di comprendere velocemente i presupposti teorici che sottendono a una buona, immediata e continua pratica alla programmazione in C (linguaggio di base) prima e C++ (linguaggio oggetto specifico di questo corso) dopo.

Le dispense, quindi, non intendono sostituirsi completamente ai libri di testo consigliati nel programma dell'insegnamento, ma rappresentano sicuramente un valido strumento per tenersi al passo con gli argomenti trattati a lezione e un ottimo riferimento per un rapido test del livello di apprendimento durante la frequenza o in vista della prova scritta di esame.

ESERCIZIO 1: *dichiarazione di variabile p di tipo puntatore a intero e suo utilizzo.*

```c
#include <stdio.h>

int main()
{
        int a;  //Inizio: dichiaro la variabile intera a
        int *p;//Inizio: dichiaro la variabile p di tipo puntatore a variabile intera
        a = 5;  //assegno un valore alla variabile intera a, ovvero attualizzo il valore di a
        p = &a;//assegno un valore a p, ovvero le assegno l'indirizzo di memoria di a
        printf("  a vale %d ", a);      // stampa il valore corrente di a
        printf("  a vale %d ", *p);     // stampa di nuovo il valore corrente di a,
                                //infatti a e *p sono in alias
        return 0;
}
```

entrambe le printf fanno stampare a video

" *a vale 5* ";

verificare che sia possibile anche stampare il valore di p a video, tale numero essendo un indirizzo potrà assumere un valore diverso per ognuno di voi, ma sicuramente sarà un numero intero multiplo di 4.

ESERCIZIO 2: *sono presentate 2 versioni, entrambi i sorgenti sono sintatticamente corretti, ma solo il secondo è la soluzione al problema che viene proposto.*

Scambio di due variabili: sintatticamente corretto

```c
#include <stdio.h>

void scambia(int, int);

int main()
{
	//Inizio: dichiaro le variabili
	int a, b;
	//Prima della chiamata: inizializzo le
	//variabili
	a = 3;
	b = 5;
	//Stampo il contenuto delle variabili a e b
	printf("%d,%d\n", a, b);

	//Chiamo la funzione scambia passando i
	//valori attuali delle variabili
	scambia(a, b);
	//Osservazione: la chiamata precedente è
	//equivalente alla seguente chiamata:
	//scambia(3,5)

	//Stampo il contenuto delle variabili a e b
	printf("%d,%d\n", a, b);
	//ATTENZIONE: Mi accorgo che il valore delle
	//variabili non è stato scambiato

	return 0;
}

void scambia(int A, int B)
{
	//Durante la chiamata: in A e B ci sono
	//i valori (copie) di a e b

	//Effettuo lo scambio delle variabili
	//usando una variabile temporanea TEMP
	//(locale alla funzione scambia)
	int temp = A;
	A = B;
	B = temp;
}
```

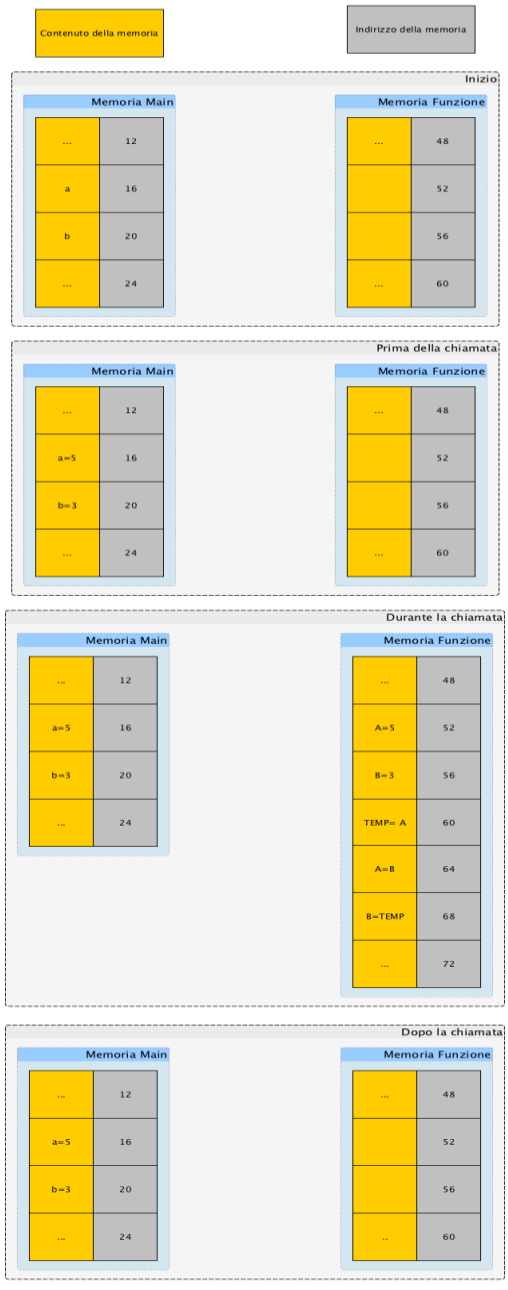

Figura 1

NB: i codici riportati in queste dispense non intendono presentare strategie mirate a risolvere alcuni problemi di mancata visualizzazione, per questo documentatevi se necessario sull'utilizzo di system("PAUSE")o altre soluzioni di vostro gradimento quali getchar().

Scambio di due variabili attraverso l'uso delle variabili puntatore.

```c
#include <stdio.h>

void scambia(int*, int*);

int main()
{
	//Inizio: dichiaro le variabili
	int a, b;
	//Prima della chiamata: inizializzo le
	//variabili
	a = 3;
	b = 5;

	//Stampo il contenuto delle variabili a e b
	printf("%d,%d\n", a, b);

	//Chiamo la funzione scambia passando come
	//argomento gli indirizzi di memoria delle
	//variabili a e b, quindi &a e &b
	scambia(&a, &b);
	//Osservazione: la chiamata precedente è
	//equivalente alla seguente chiamata:
	//	scambia(16,20)

	//Stampo il contenuto delle variabili a e b
	printf("%d,%d\n", a, b);
	//ATTENZIONE: Mi accorgo che in questo caso
	//il valore delle variabili è stato scambiato

	return 0;
}

void scambia(int*A, int*B)
{
	//Durante la chiamata: A e B contengono gli
	//indirizzi delle aree di memoria in cui sono
	//memorizzate le variabili a e b
	//rispettivamente

	//Alla variabile temp assegno il valore
	//contenuto nell'area di memoria il cui
	//indirizzo è memorizzato in A
	int temp = *A;

	//Nell'area di memoria "puntata da A" salvo il
	//contenuto dell'area di memoria puntata da B
	*A = *B;
	//Nell'area di memoria "puntata da B" copio il
	//contenuto della variabile temp.
	*B = temp;
}
```

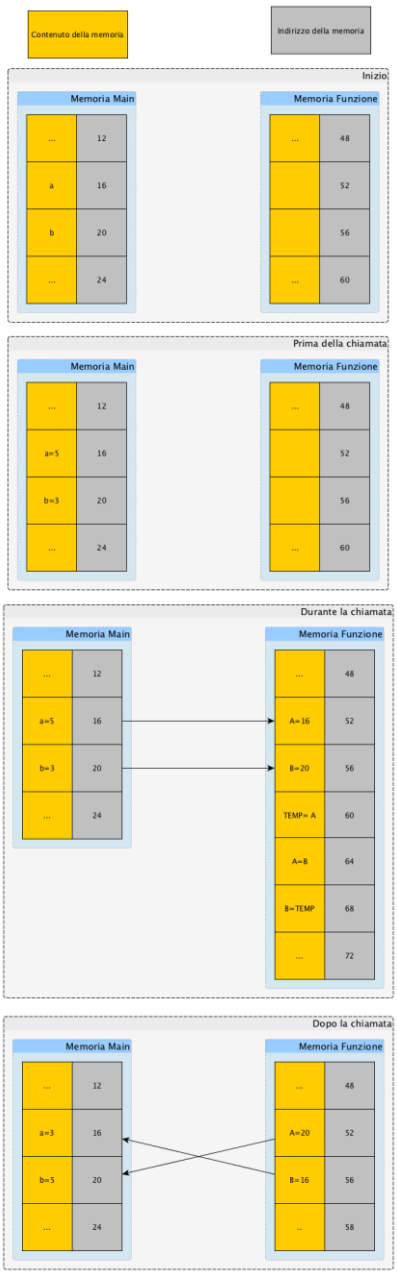

Figura 2

Array (vettori o matrici per esempio), stringhe (vettori di carattere con un ultimo elemento riservato per carattere di fine stringa) e relativi puntatori.

```c
#include <stdio.h>

int main() {

	int V[10]; //dichiaro un vettore con max dimensione statica pari a 10
// risultano accessibili le componenti da V[0] a V[9]
// V è anche l'indirizzo del vettore, è sinonimo di &V[0] quindi è di tipo int *
// V rappresenta in questo caso l'indirizzo di memoria di una sequenza di 40 bytes adiacenti
	return 0;
}
```

```c
int main() {

	int M[20][30];
//dichiaro una matrice con max dimensione statica di righe pari a 20 e colonne pari a 30
// risultano accessibili le componenti da M[0][0] a M[19][29]
// M è anche l'indirizzo della matrice vettore, è sinonimo di &M[0][0]
// M in questo caso è l'indirizzo di memoria di una sequenza di 2400 bytes adiacenti
// M è di tipo int(*)[30]
	return 0;
}
```

In particolare **le matrici sono allocate per righe**. Nel caso di una matrice di caratteri M[20][30], in memoria verranno allocati complessivamente 600 byte:

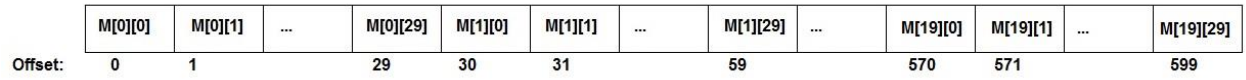

Figura 3

Stringhe

```
int main()
{
        char parola[4]; //dichiaro una stringa di max dimensione statica pari a 4
        scanf("%s", parola);
        //magari se provo ad acquisire la parola ciao
        printf("%s", parola);
        //verrà stampato ciao
        return 0;
}
```

In parola[3] è memorizzato il carattere ASCII del terminatore di stringa ovvero ' \0' , in parola[0] il carattere ASCII della lettera c ovvero 'c'.

Funzioni da libreria

Le più comuni funzioni per il trattamento delle stringhe, i cui prototopi sono nella libreria **string.h**, sono:

strcpy(s1, s2)	copia il contenuto della stringa s2 in s1. La stringa s1 deve essere abbastanza grande da contenere il contenuto della stringa s2; *strncpy(s1, s2, n)* copia i primi n caratteri della stringa s2 in s1.
strcat(s1, s2)	aggiunge la stringa s2 alla fine della stringa s1 (concatenamento). La stringa s2 rimane invariata mentre s1 conterrà la stringa risultante dalla la concatenazione di s1 e s2. Entrambe le stringhe devono terminare con il carattere NULL e il risultato sarà un'unica stringa con il carattere terminatore NULL alla fine.
strcmp(s1, s2)	confronta due stringhe e restituisce 0 se sono uguali. Ritorna un valore maggiore di zero se, lessicograficamente (ovvero, in base all'ordine del dizionario), s1 è maggiore di s2. Se, invece, s1 è minore di s2, lessicograficamente, ritorna un valore minore di zero.
strlen(s)	restituisce la lunghezza della stringa s.

Tabella 1

ESERCIZIO 3: passaggio di una matrice a una funzione e manipolazione

```c
#include <stdio.h>
#define MACRO_MAX_COLONNE 10

/*dichiarazione del prototipo della funzione*/
void funzione(int, int, int(*)[MACRO_MAX_COLONNE]);
                                        // il nuovo tipo int(*)[MACRO_MAX_COLONNE] consente di
                                        // informare il compilatore che il terzo argomento sarà un
                                        //puntatore al primo elemento di una matrice con numero
                                        //di colonne pari a MACRO_MAX_COLONNE


int main() {

        //variabili locali al main
        int A[10][10];//nello scope del main, il compilatore alloca 400 byte per la
         //matrice A
        int i, j;
        int R, C;
        R = 2; C = 3;

        for (int i = 0; i < R; i++)
                for (int j = 0; j< C; j++)
                        A[i][j] = 0;


        printf("Stampa della matrice nulla\n");
        for (i = 0; i < R; i++) {
                for (j = 0; j < C; j++)
                        printf("%d ", A[i][j]);

                printf("\n");
        }


        /*chiamata alla funzione*/
        funzione(R, C, A);      //R, C, A sono i parametri attuali della funzione;
        //nel terzo argomento della funzione, il valore attuale che viene passato è
        //l'indirizzo del primo elemento della matrice A


        printf("Stampa della matrice modificata dalla chiamata alla 'funzione()'\n");

for (i = 0; i < R; i++)
```

```
        {
                for (j = 0; j < C; j++)
                        printf("%d ", A[i][j]);

                printf("\n");
        }

        return 0;
}



/*definizione della funzione*/
void funzione(int A, int B, int (*C)[MACRO_MAX_COLONNE])
{                               //il terzo argomento è un puntatore a un array di lunghezza 10;
                                //il compilatore alloca nello scope della funzione
                                //400 byte per la matrice C, in cui per es. l'elemento (2,1)
                                //ha come indirizzo: &C[0][0]+4*offset
                                //con offset = MACRO_MAX_COLONNE*2+1
        int i, j;
        for (i = 0; i < A; i++)
                for (j = 0; j< B; j++)
                        C[i][j] = 1;

        /*stampa di verifica degli indirizzi in memoria, nello scope locale*/
        int offset = MACRO_MAX_COLONNE * 2 + 1;
        int indirizzo = (int)&C[0][0] + 4*offset;
        printf("\nIndirizzo di &C[0][0] : %d\n", &C[0][0]);
        //I valori di 'indirizzo' e di '&C[2][1]' devono coincidere
        printf("\nIndirizzo di &C[0][0]+4*offset : %d\n", indirizzo);
        printf("\nIndirizzo di &C[2][1] : %d\n", &C[2][1]);
        printf("\n4*Offset = %d\n\n", 4*offset);

}
```

I dati strutturati

Fino a questo momento sono stati introdotti ed usati solamente i tipi di dati fondamentali (int, char, float, double), ossia tipi di dati **built-in**, già presenti all'interno del compilatore. A partire da questi è possibile costruire nuove tipologie di dati, definiti **dati strutturati**.

Un tipo di dato strutturato viene dichiarato tra le inclusioni delle librerie ed il main e la parola chiave per dichiararlo è struct, seguita dal nome identificativo del tipo di dato. Successivamente, all'interno di parentesi graffe, si definiscono le variabili interne della struttura, ovvero i campi che compongono il dato strutturato. Si noti come dopo la chiusura delle parentesi graffe va inserito il punto e virgola perché la dichiarazione di una struttura è un'istruzione.

```
#include <stdio.h>
struct studente
{
        char cognome[20];
        char nome[20];
        char matricola[8];
        int voto;
};

int main()
{
        ...
        return 0;
}
```

Da questo momento in poi, sarà possibile dichiarare una variabile di tipo studente:

```
struct studente alunno;
```

Per accedere al campo di una variabile di tipo struttura si fa uso del "." posto tra il nome della variabile strutturata e quello del campo a cui si vuole accedere.

Come si può intuire, con quest'istruzione viene definita una variabile alunno composta da 52 byte consecutivi in memoria (20 per "nome", 20 per "cognome", 8 per "matricola", 4 per "voto").

Il seguente esempio mostra quanto detto finora, ovvero la dichiarazione di un nuovo tipo di dato del tipo struct e relative funzioni di manipolazione.

ESERCIZIO 4: *si voglia progettare e implementare in linguaggio C un software di gestione che consenta ad un professore di visualizzare i risultati di una prova di esame, con diverse possibilità di ordinamento e visibilità dei dati di ogni singolo studente. Si supponga che il prof abbia richiesto la prenotazione all'esame e che in totale si siano prenotati 100 studenti. Il giorno dello svolgimento agli studenti è richiesto di rendere noto: Cognome, Nome e Matricola. Al termine dell'esame il prof raccoglie i compiti degli studenti che decidono di consegnare l'elaborato. Dopo aver corretto ed assegnato un voto da 0 a 30 a ciascun compito, il prof memorizzerà mediante il software in questione, per ciascuno studente, Cognome, Nome, Matricola e Voto. Infine si richiede, per esempio, di consentire al prof di stampare a video in ordine di voto decrescente i campi Matricola e Voto di ciascuno studente.*

```c
#include <stdio.h>

/*il nuovo tipo di dato strutturato va dichiarato prima di qualunque funzione*/

struct studente    //modellazione del nuovo tipo di dato
{
	char cognome[20];  //sovrastimiamo la lunghezza del cognome
	char nome[20];     //sovrastimiamo la lunghezza del nome
	char matricola[8]; //n. di elementi arrotondato per ecc. al mult.
	                   //di 4 più vicino al n. di cifre della matr.
	int voto;    //varierà tra 0 e 30
};//ricordare il ';'


  /*dichiarazione funzioni*/
void leggi(int, struct studente*);
void ordina(int, struct studente*);
void stampa(int, struct studente*);
void scambia(struct studente*, struct studente*);

int main()
{
	int dim;     //n. di studenti che hanno effettivamente consegnato
	struct studente classe[100];    //la classe d'esame è un vettore di
	                                //lunghezza sovrastimata in base al
	                                //n. di studenti prenotati,
	                                //tipizzato col nuovo tipo di dato

	/*quanti studenti hanno consegnato?*/
	do
	{
		printf("Inserisci il numero di studenti\n");
		scanf("%d", &dim);
	} while (dim>100); //al più 100 studenti
```

```
	/*chiamata alla funzione leggi*/
	leggi(dim, classe);

	printf("Gli studenti che compongono la classe d'esame sono:\n");
	/*chiamata alla funzione stampa*/
	stampa(dim, classe);

	/*chiamata alla funzione ordina*/
	ordina(dim, classe);

	printf("\nStampa dei risultati:\n");
	/*chiamata alla funzione stampa*/
	stampa(dim, classe);
	return 0;
}

/*definizione della funzione leggi*/
void leggi(int A, struct studente *B)
{
	int i;

	for (i = 0; i<A; i++)
	{
		printf("\n\nInserisci i dati dello studente n. %d", i + 1);

		printf("\nCognome: ");
		scanf("%s", B[i].cognome); //non c'è bisogno della '&' perchè
					//'B[i].cognome' è l'indirizzo del
					//primo elemento della stringa
					//cognome (il dichiaratore di tipo
					//'%s' evita di farci enumerare i
					//caratteri)
		printf("\nNome: ");
		scanf("%s", B[i].nome); //si ricordi che l'ultimo carattere è
				//riservato al carattere di fine stringa


		printf("\nMatricola: ");
		scanf("%s", B[i].matricola); //al più 7 cifre

		do
		{
			printf("\nVoto (da 0 a 30): ");
			scanf("%d", &B[i].voto); //ricordarsi la '&', è un int!
		} while (B[i].voto <0 || B[i].voto >30);
	}
}

/*definizione della funzione stampa*/
void stampa(int A, struct studente *B)
{
	int i;
	printf("Matr.\t\tCognome\t\tNome\t\tVoto\n");
	for (i = 0; i < A; i++)
printf("%s\t\t%s\t\t%s\t\t%d\n", B[i].matricola, B[i].cognome, B[i].nome, B[i].voto);

	//non si usa la '&' perchè deve essere stampato
	//il valore di 'B[i].voto' e non l'indirizzo
}
```

```
/*definizione della funzione ordina - Algoritmo SELECTION SORT*/
void ordina(int A, struct studente *B)
{
	int imax, i, j;

	//2 cicli for annidati
	for (i = 0; i <A - 1; i++) {	//questo ciclo fa sempre A-1 passi

		imax = i;

		//fissato l'indice i, si individua l'indice j dell'elemento più
		//grande dell'elemento di indice i (con j che varia tra i+1 e A-1)
		for (j = i + 1; j<A; j++)
			if (B[j].voto>B[imax].voto) imax = j;
		//si aggiorna il valore attuale di imax
		//quando è verificata la condizione (l'if
		//contiene il solo ramo della verità)

		/*chiamata alla funzione scambia*/
		//si scambiano gli elementi di indice i e imax
		scambia(&B[i], &B[imax]); //il Selection Sort effettua
					//al più A-1 scambi
	}

}


/*definizione della funzione scambia*/
void scambia(struct studente *A, struct studente *B)
{
	struct studente temp; //var. locale di tipo struct studente

	temp = *A;	//assegno alla variabile 'temp' il valore
			//contenuto nell'area di memoria il cui
			//indirizzo è memorizzato in A

	*A = *B;	//Nell'area di memoria "puntata da A" salvo il
			//contenuto dell'area di memoria puntata da B

	*B = temp;	//Nell'area di memoria "puntata da B" copio il
			//contenuto della variabile 'temp'
}
```

A margine della soluzione dell'esercizio, si precisa che è stato possibile fare riferimento all'intera struttura anziché ai singoli membri per farne una copia. Si osservi il seguente codice:

```
struct studente alunno1, alunno2;
...
alunno2 = alunno1;
```

Si noti che l'assegnazione precedente è equivalente al seguente gruppo di istruzioni:

```
struct studente alunno1,alunno2;
int i;

...

for(i=0;i<20;i++)
{
	alunno2.cognome[i]=alunno1.cognome[i];
```

```
            alunno2.nome[i]=alunno1.nome[i];
        }
        ...
        alunno2.voto=alunno1.voto;
```

Dunque tutto il contenuto della variabile alunno2, è stato copiato all'interno della variabile alunno1.

Impatto sulla memoria

Facendo riferimento all'esercizio precedente, consideriamo come viene riservata la memoria per una classe d'esame di n. 3 studenti (Figura 5).

```
struct studente classe[3];
```

Nello scope del main sarà riservata un'area di memoria per il vettore di 3 elementi di tipo struct studente. Si supponga di utilizzare una macchina in cui la memoria centrale sia composto da registri di parole da 32 bit (4 byte).

Per ciascuno studente l'area di memoria consentirà di memorizzare **52 byte**:

20 byte per il campo '`classe[0].cognome`'
20 byte per il campo '`classe[0].nome`'
8 byte per il campo '`classe[0].matricola`'
4 byte per il campo '`classe[0].voto`'

Per quanto riguarda l'indirizzamento in memoria, si noti che l'indirizzo assoluto del vettore "`classe`" coincide con:

- l'indirizzo assoluto del 1° studente, "`classe[0]`";
- l'indirizzo assoluto del campo "cognome" del 1° studente, "`classe[0].cognome`";
- l'indirizzo assoluto del 1° carattere del cognome del 1° studente, "`classe[0].cognome[0]`";.

Infine la chiamata alla funzione "`scambia(&B[i],&B[j])`", consente di effettuare lo scambio delle due intere aree di memoria di 52 byte, avendo passato alla funzione stessa unicamente l'indirizzo assoluto del primo byte delle due strutture dati.

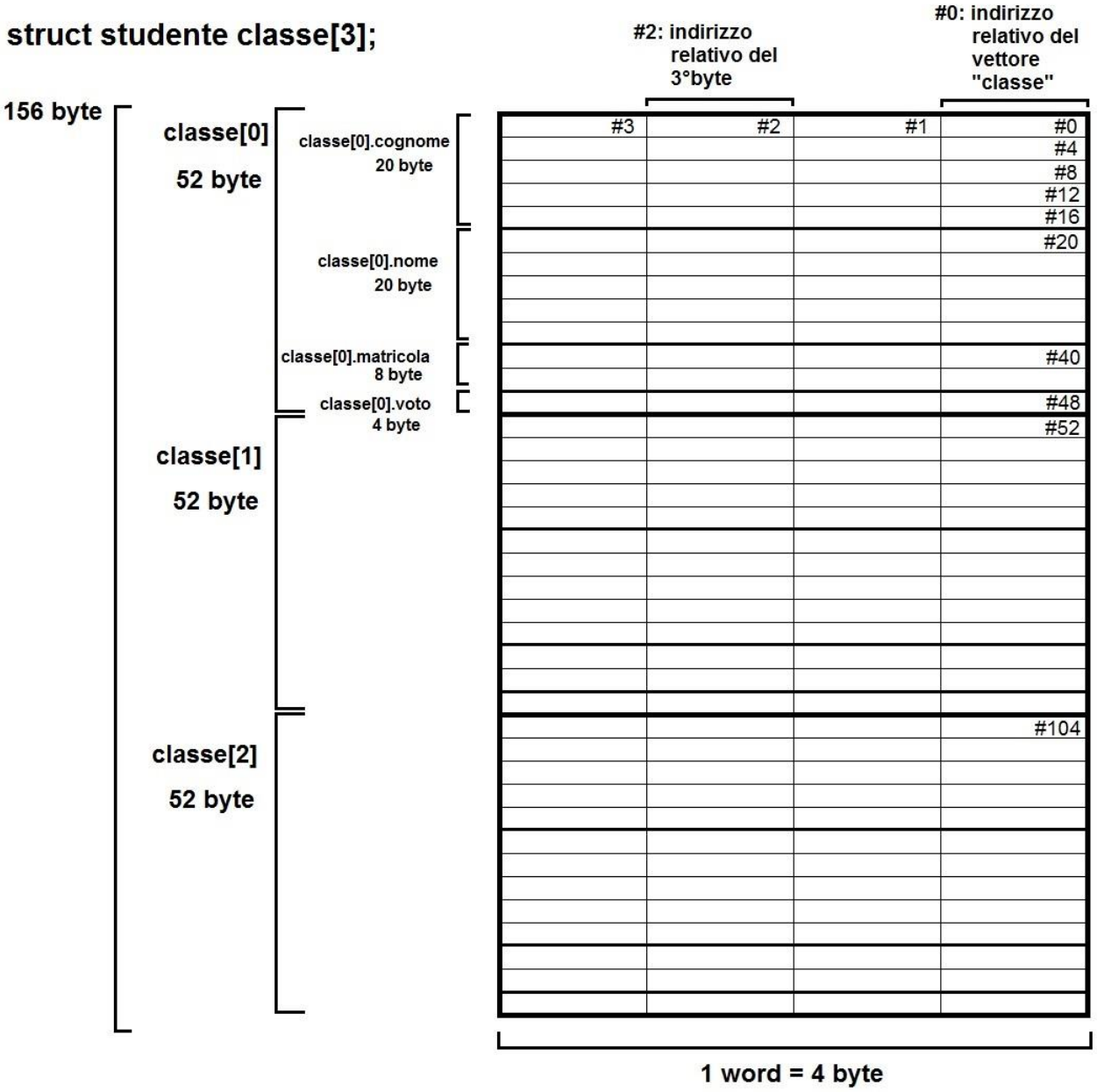
ALLOCAZIONE DELLA MEMORIA: SCOPE DEL "MAIN"
struct studente classe[3];
156 byte
classe[0]
52 byte
classe[0].cognome
20 byte
classe[0].nome
20 byte
classe[0].matricola
8 byte
classe[0].voto
4 byte
classe[1]
52 byte
classe[2]
52 byte
#2: indirizzo relativo del 3°byte
#0: indirizzo relativo del vettore "classe"
#3
#2
#1
#0
#4
#8
#12
#16
#20
#40
#48
#52
#104
1 word = 4 byte

Figura 4