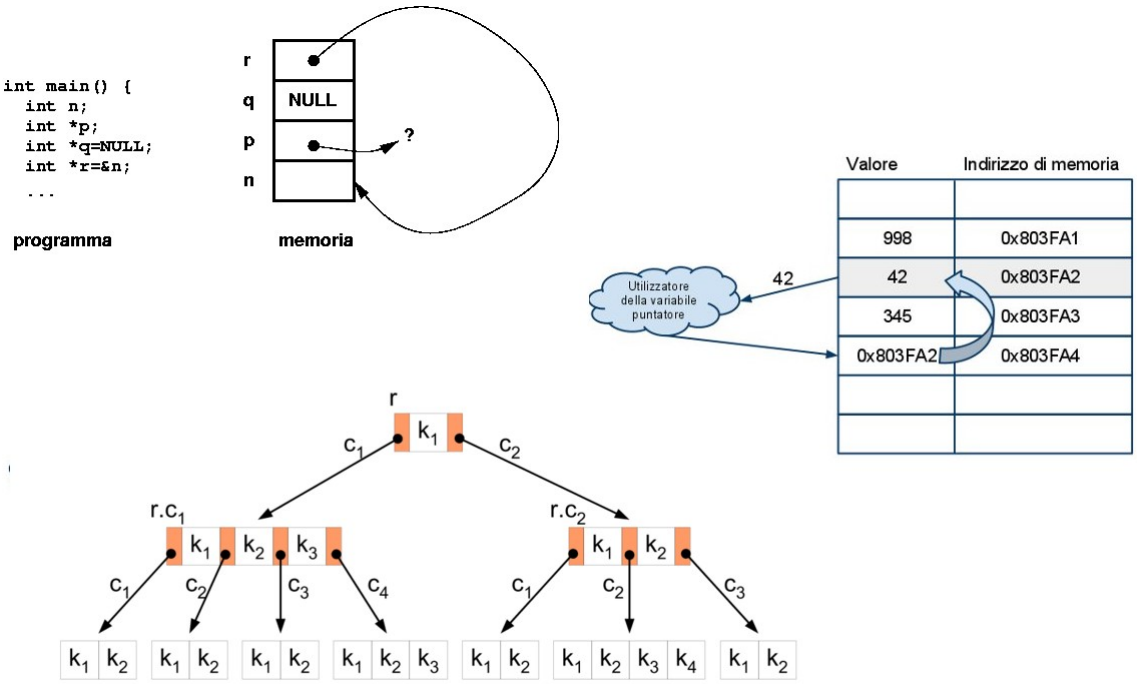
ITIS-LS "Francesco Giordani" Caserta

prof. Ennio Ranucci

a.s. 2019-2020

Semplici esercitazioni che utilizzano Strutture concatenate

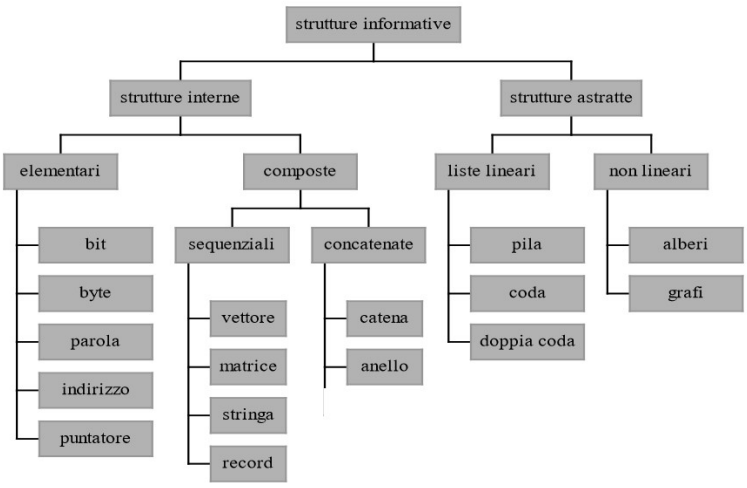
Puntatori



***La lista** è un insieme di dati in cui il primo non ha predecessore, l'ultimo non ha successore, tutti gli altri hanno predecessor e successore.*
***La Lista** è una struttura astratta che possiamo implementare con un array c++ e quindi con una struttura statica (celle contigue che vengono riservate a tempo di compilazione) oppure con una struttura dinamica (celle collegate da puntatori che vengono riservate a tempo di esecuzione).*
Gestione delle liste:
*Il termine **FIFO** è l'acronimo inglese di First In First Out e rappresenta un metodo per gestire una fila di oggetti: "primo ad entrare, primo ad uscire".*
Il metodo FIFO rappresenta la modalità di immagazzinamento di oggetti fisici in cui il primo oggetto introdotto è il primo ad uscire. Un esempio di immagine rappresentativa del concetto è quella di un tubo con una estremità da cui entrano gli oggetti e l'altra dalla quale escono.
L'ordine di uscita è uguale a quello di entrata.
Si può esemplificare con un dispenser di prodotti in un supermercato, in cui gli articoli vengono introdotti dall'alto ed il cliente li preleva dal basso, permettendo la rotazione di tutti i prodotti; o più semplicemente una serie di persone che disposte in fila indiana attendono di essere servite al bancone di una biglietteria.
*Il metodo FIFO si contrappone al metodo **LIFO** (Last In-First Out "ultimo arrivato primo uscito") in cui è l'ultimo oggetto inserito ad essere estratto per primo. Ad esempio una catasta di cassette disposte una sopra l'altra.*

Ai due livelli di analisi e programmazione corrispondono anche due modalità di rappresentare i dati del problema. Durante l'analisi, il problema viene affrontato e risolto mediante un algoritmo che prescinde dalla natura dell'elaboratore e che agisce sui dati del problema. Tali dati dovranno essere rappresentati in una struttura, detta struttura astratta perché indipendente all'elaboratore, che possa essere trattata dall'algoritmo nel modo più efficiente possibile. Le strutture astratte sono rappresentazioni dei dati di un problema che rispecchiano le proprietà dei dati e le relazioni usate nella stesura dell'algoritmo risolutivo. In generale si parla di tipo di dati riferendosi all'insieme costituito da una struttura e dalle operazioni definite su di essa.

Durante la fase di programmazione, l'algoritmo viene tradotto, tramite un opportune linguaggio di programmazione, in una forma eseguibile dalla macchina, e così anche le strutture astratte dei dati dovranno essere trasformate in strutture interne, rappresentabili nella memoria della macchina utilizzata. Per ogni tipo di dati astratto è utile scegliere la struttura interna che si presta meglio alla sua rappresentazione.

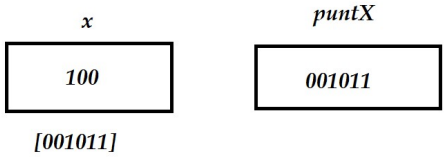


*Ad esempio la variabile come **struttura astratta** può definirsi come una tripla: identificatore, valore attuale, tipo; come **struttura interna o concreta** è una locazione di memoria il cui indirizzo base è dato dall'identificatore e la sua lunghezza è data dal tipo, il valore attuale è la stringa binaria in essa rappresentata. Il tipo restituisce due informazioni: la lunghezza della locazione e la chiave per interpretare la stringa di 0 ed 1 contenuta nella locazione stessa.*

Cosa è un puntatore?

Un puntatore è una variabile che contiene un indirizzo di memoria in cui è memorizzata un'altra variabile, è, cioè, una variabile che contiene l'indirizzo di un'altra variabile

I puntatori sono "type bound" cioè ad ogni puntatore è associato il tipo a cui il puntatore si riferisce



Puntatori

Una variabile puntatore è una variabile che contiene l'indirizzo di memoria di un'altra variabile. Ad esempio, si supponga di avere una variabile intera chiamata x ed un'altra variabile (quella puntatore appunto) chiamata puntX che è in grado di contenere l'indirizzo di una variabile di tipo int. In C++, per conoscere l'indirizzo di una variabile, è sufficiente far precedere al nome della variabile l'operatore &. La sintassi per assegnare l'indirizzo di una variabile ad un'altra variabile è la seguente:

puntX = & x;

In genere, quindi, una variabile che contiene un indirizzo, come ad esempio puntX, viene chiamata variabile puntatore o, più semplicemente, puntatore.

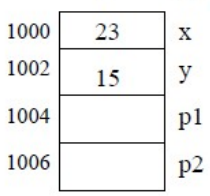
*Per accedere al contenuto della cella il cui indirizzo è memorizzato in puntX è sufficiente far precedere alla variabile puntatore il carattere asterisco **

*puntX = & x; *puntX = 20; //il valore 20 viene assegnato alla cella x*

*x = 20; *puntX = 20; sono istruzioni equivalenti*

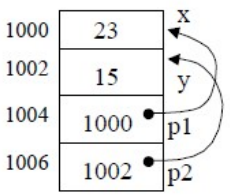
Si supponga di avere la seguente :

```
int x,y;
int *p1,*p2;
x = 23;
y = 15;
```



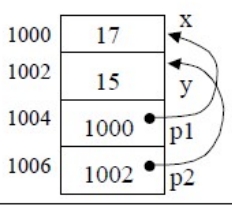
Dopo le operazioni di assegnazione:

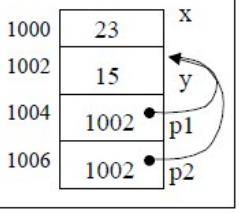
p1 = &x;
p2 = &y;



Le espressioni:

*p1 = 17;
p1 = p2;
provocano le situazioni che seguono:





L'indirizzo di una variabile non basta per poter leggere il suo contenuto

E' necessario sapere quanti bit bisogna prendere in considerazione a partire dall'indirizzo indicato e cosa rappresentano (un intero, un numero reale, un catattere, una stringa, ecc.).

Queste due informazioni sono contenute nel **TIPO** della variabile.

Quindi quando dichiariamo un puntatore bisogna indicare il tipo di variabile a cui punta, ad esempio: **int *ptr1;**

Prendiamo confidenza con le variabili puntatore

PUNT1

```
#include <iostream>
using namespace std;
int main()
{
 int *ptr1;
 ptr1 = 0; //indica che ptr1 non punta a nulla
 if (ptr1 == 0)
 cout << "Il puntatore non e' inizializzato"<<endl;
 int conta = 7;
 int *contaInd;
 contaInd=&conta; //assegna a contaInd l'indirizzo della variabile intera conta
 cout<<"valore della variabile conta: "<<conta<< endl;
 cout<<"indirizzo di conta: "<<contaInd<< endl;
 cout<<"valore della variabile conta ottenuto mediante il suo indirizzo: "<<*contaInd;
 return 0;
}
```

PUNT2

```
#include <iostream>
using namespace std;
int main()
{
   int x,y;
   int *p1,*p2;
   x = 23;
   y = 15;
   p1 = &x;
   p2 = &y;
   cout<<*p1<<" "<<*p2;
   return 0;
}
```

Allocazione dinamica

I puntatori permettono di gestire la memoria dinamicamente, cioè allocare (occupare) spazio per una certa variabile solo quando questa deve essere usata

L'operatore new alloca un nuovo oggetto

- ☞prende il nome del tipo dell'oggetto come operando
- ☞restituisce una referenza all'oggetto
- ☞lo spazio è ricavato nell'area dati dinamica (*heap*)
- ☞pointer = new int; /* crea un oggetto di tipo intero; pointer punta a tale oggetto */

La funzione standard utilizzate in C per l'allocazione dinamica è malloc(). Essa può essere anche utilizzate in C++

PUNT3

```
#include <iostream>
using namespace std;
int main()
{
   int i = 10, *p1, *p2;
   p1 = &i;
   cout << "valore contenuto all'indirizzo p1 della memoria di lavoro: "<<*p1 << endl;
   p2 = new int; //creazione di uno spazio della memoria heap di tipo int
   *p2 = *p1;
   cout <<"valore contenuto all'indirizzo p2 della memoria heap:"<<*p2 << endl;
   *p1 = 5;
   cout << "valore dello spazio di memoria puntato da p1: "<< *p1 <<endl;
   cout << "valore dello spazio di memoria puntato da p2: " << *p2 << endl;
   return 0;
}
```

PUNT4

```
#include <iostream>
using namespace std;
void swap(int x, int y)
{
 int temp;
 temp = x;
 x = y;
 y = temp;
}


void swapIndirizzo(int *x_ptr, int *y_ptr)

{

 int temp;

 temp = *x_ptr;

 *x_ptr = *y_ptr;

 *y_ptr = temp;

}

int main()
{
   int n1=3,n2=5;
   swap(n1,n2);
   cout<<n1<<" "<<n2<<endl;
/*
I valori di n1 ed n2 non risultano scambiati. Questo accade perché la funzione swap agisce solo
su una copia delle variabili n1 e n2 Per far sì che la funzione agisca sulle variabili stesse e non su
delle copie occorre passare gli indirizzi
*/
   swapIndirizzo(&n1,&n2);
   cout<<n1<<" "<<n2<<endl;
   return 0;
}
```

Gestione statica, automatica e dinamica della memoria

Allocazione dinamica

Con allocazione dinamica si intende l'occupazione della memoria per l'utilizzo di un programma durante la propria esecuzione. Questo metodo è utilizzato per distribuire il possesso di limitate quantità di memoria tra varie porzioni di dati e codice. Un oggetto allocato dinamicamente rimane tale fintanto che non viene deallocato esplicitamente, dal programmatore o da un Garbage Collector.

Heap e Stack

A livello logico, è possibile immaginare l'allocazione dinamica come la gestione di due grandi aree di memoria: heap e stack. Mentre lo stack è paragonabile ad una pila che cresce dal basso verso l'alto contenente informazioni necessarie all'esecuzione dei metodi e i dati risultanti dalla loro invocazione, lo heap memorizza gli oggetti istanziati dal programma e rappresentati da puntatori presenti nello stack. L'area dello heap inutilizzata sarà resa di nuovo utilizzabile appunto dal Garbage Collector.

Le variabili dinamiche possono essere individuate solo tramite un puntatore. Se non conosciamo il puntatore non possiamo più accedere alla variabile stessa.

L'area di memoria dinamica, denominata heap, consente di allocare spazi di memoria la cui dimensione è nota solo a runtime (in fase di esecuzione del programma) e per questo, si dice, dinamicamente. Una variabile viene allocata nell'heap utilizzando opportune istruzioni messe a disposizione dal linguaggio di programmazione. Una variabile allocata dinamicamente ha la caratteristica di avere un tempo di vita che non è legato a quello delle funzioni. In particolare, non è legato al tempo di esecuzione della funzione in cui essa viene creata, nel senso che sopravvive anche dopo che la la funzione termina. Ciò consente di creare una variabile dinamica in una funzione, di utilizzarla in altre funzioni e di distruggerla (deallocarla esplicitamente con le istruzioni opportune) in una funzione ancora diversa.

Allocazione statica

Con allocazione statica della memoria si intende l'occupazione di memoria durante la fase di compilazione. Generalmente, le variabili così gestite possiedono un identificatore speciale, come la parola "static" in C e Java.

int a[5]; int b[2][3]; char c[4]; int somma;

Le variabili dichiarate in questo modo non possono variare le loro caratteristiche a tempo di esecuzione. Ad esempio, non possiamo modificare l'array a tempo di esecuzione in modo che questo non sia costituito da 5 celle adiacenti.

Allocazione automatica

Le variabili automatiche sono variabili locali all'interno di un blocco di istruzioni. Esse sono allocate nello stack quando si entra in quel blocco di codice; parallelamente vengono distrutte quando si esce dal blocco stesso. Alla variabile, in questo caso, viene assegnato un indirizzo di memoria solo quando è richiesto, ovvero subito dopo la chiamata alla funzione. La corrispondenza variabile-indirizzo è definita come "Scope di memorizzazione" ed è verificata solo localmente al blocco di codice e non all'intero programma.

Per l'allocazione dinamica di una variabile il linguaggio C++ mette a disposizione due sintassi.

La prima consiste nell'utilizzo della funzione seguente: void* malloc(size_t numero_bytes); Essa alloca nella area dell'heap un blocco di memoria costituito da un numero di bytes pari al valore del parametro numero_bytes passato e restituisce al chiamante un puntatore al primo byte del blocco allocato. Si noti che questo puntatore è generico (void*) e che su di esso bisogna sempre eseguire un casting esplicito verso il tipo della variabile che si vuole allocare.

Esempio

ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es0*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Code::Blocks Release 17.12 rev 11256*
Obiettivo didattico:
Utilizzo della funzione malloc per la gestione dinamica della memoria
Obiettivo del programma:
Caricare nella memoria dinamica un numero n di interi a tempo di esecuzione

```
#include <iostream>
#include <stdlib.h> /* malloc, free, rand */
using namespace std;
int main()
 {
   int n, num;
   cout << "Quanti sono gli interi da caricare?" << endl;
   cin >> n;
   //Allocazione dinamica di un blocco di memoria pari a n int
   int* v = (int*) malloc(sizeof(int)*n);
   if (v==NULL) exit(1); //Allocazione non andata a buon fine!
   for(int i=0; i<n; i++)
     {
       cout<<"Inserisci l'intero n. "<<i+1<<": ";
       cin>>v[i];
     }
   cout<<"Sono stati memorizzati i seguenti "<<n<<" interi:";
   for(int i=0; i<n; i++)        cout<<" " << v[i];
   cout<<endl;
   //Dealloca l'intero blocco di memoria puntato da v
   free(v);
   return 0;
}
```

In C++ un blocco di memoria dell'heap può essere riallocato modificandone la dimensione utilizzando la seguente funzione di libreria: void* realloc (void* ptr, size_t numero_bytes);

La seconda possibilità consiste nell'utilizzo dell'operatore new.

Esempio

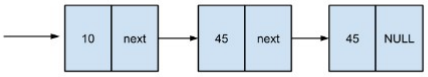
ITIS-LS "Francesco Giordani" Caserta
Anno scolastico: *2019/2020*
Classe 3^ sez.B spec. Informatica e telecomunicazioni
Data:
Numero progressivo dell'esercizio: *es1*
Versione: *1.0*
Programmatore/i:
Sistema Operativo: *Windows 10*
Compilatore/Interprete: *Code::Blocks Release 17.12 rev 11256*
Obiettivo didattico:
Utilizzo della funzione new per la gestione dinamica della memoria
Obiettivo del programma:
Generare a tempo di esecuzione una stringa composta da n caratteri casuali nella memoria dinamica

```
#include <iostream>
#include <stdlib.h> // srand, rand
#include <time.h>   // time
using namespace std;
int main()
{
  char car;
  int n,i;
  cout<<"Quanti caratteri deve essere lunga la stringa casuale? ";
  cin>>n;
  /*Allocazione dinamica di un blocco di memoria sufficiente ad  accogliere n char, più 1 per il carattere di fine stringa*/
  char* s = (char*) new char[n+1];
  if(s==NULL) exit(1); //allocazione dinamica non andata a buon fine!
  srand (time(NULL)); //inizializza il seme per la funzione rand()
  for(i=0; i<n; i++)
    {
     //generea una lettera casuale e l'aggiunge alla stringa s
     car = rand()%26 + 'A';
     s[i]=car;
    }
  s[i]='\0';
  cout<<"La stringa casuale di "<<n<<" caratteri prodotta e': "<<s<< endl;
  //Dealloca l'intero blocco di memoria puntato da s;
  delete [] s;
  return 0;
}
```

L'operatore new richiede di specificare il tipo di variabile che si vuole allocare nell'heap (nel caso del nostro esempio un vettore di n+1 char) e restituisce il puntatore al primo byte del blocco di memoria allocato. Anche in questo caso il puntatore restituito è generico (void*) e, pertanto, su di esso bisogna sempre eseguire un casting esplicito verso il tipo della variabile che si vuole allocare.

Strutture concatenate

Una lista concatenata semplice, o linked list, è una struttura dati che consente la memorizzazione di informazioni in elementi, detti anche nodi, non contigui collegati sequenzialmente da un puntatore .



Lista concatenata semplice di 3 interi

Le liste sono una struttura dati molto usata nel''informatica perché
- memorizzano una collezione di elementi in modo non contiguo;
-consentono l'inserimento o la cancellazione di un elemento con un costo costante, in termini di complessità di calcolo;
-sono una struttura dati dinamica le cui dimensioni possono essere modificate arbitrariamente;
-possono essere sia ordinate che non ordinate;
possono essere usate per implementare molte strutture dati astratte quali: pile (stacks), code (queues), code doppie (deques), sequenze (sequences), code di priorità (priority queues) e altro ancora.

Per implementare in linguaggio C, in modo efficiente ed efficace, una lista si usano i puntatori e il tipo di dato complesso struct.
Le frecce si implementano con i puntatori e i rettangoli le struct.
Nel rettangolo si memorizza il dato di interesse, che può essere semplice o strutturato (una struct) e il puntatore all'elemento successivo. L'ultimo elemento della lista conterrà un puntatore pari a NULL.

```
typedef struct nodo{

 int  num;          // dato contenuto nel generico nodo della lista

 struct nodo *next;   // puntatore al prossimo elemento della lista

} nodoT;                    // Tipo del generico nodo della lista
```

l'implementazione del tipo della "freccia" è la seguente:

```
typedef nodoT* nodoP; // Tipo puntatore a lista

nodoP firstP = NULL;       // puntatore all'inizio della lista
```

Poiché è molto utile poter accedere rapidamente anche all'ultimo elemento della lista, può essere utile avere una variabile puntatore per questo scopo, da aggiornare in modo opportuno durante le operazioni che modificano la lista:

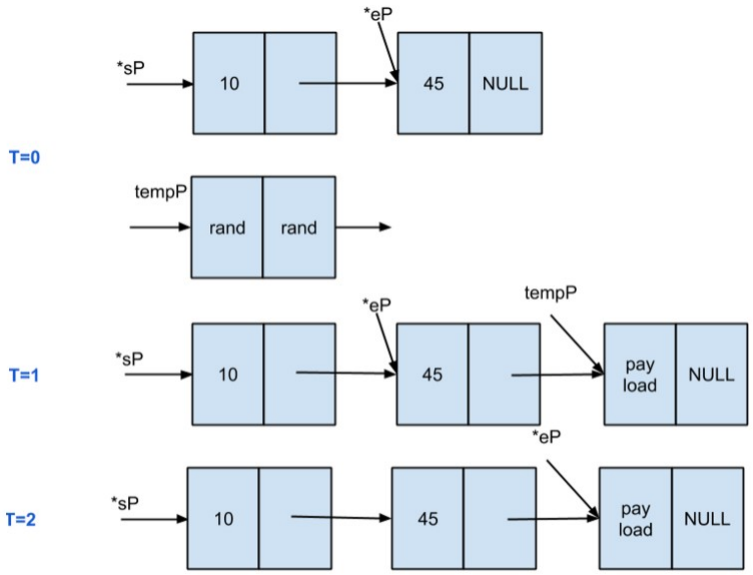
```
nodoP lastP  = NULL;       // puntatore all'ultimo nodo della lista
```

Ovviamente, in alternativa a questo approccio, si può "visitare" la lista dall'inizio fino alla fine e ottenere così il puntatore all'ultimo elemento. La scelta tra i due approcci dipende dal tipo di operazioni che devono essere fatte sulla lista.

L'implementazione in C è quindi la seguente:

```
void addnode(int pl, nodeP* sP, nodeP* eP)
{
 nodeP tempP = (nodeP)malloc(sizeof(nodeT));
 tempP->next     = NULL;
 tempP->payload  = pl;
 if (*eP==NULL)
  {
   *sP         = tempP;
   *eP         = tempP;
  } else
 {
   (*eP)->next   = tempP;
   *eP         = tempP;
 }
}
```

Si noti che la routine riceve in ingresso un intero, per il dato da memorizzare, un puntatore al puntatore inizio lista e un puntatore al puntatore fine lista. (Att: Gli ultimi due parametri sono quindi due puntatori passati by reference, e non by value.)
Esempio di esecuzione della funzione addnode al caso di una lista contenente già due nodi. Per semplicità, sono stati descritti gli stati della lista nei tre istanti di tempo da T0 a T2



In ultimo, si noti che, in questo caso, avere a disposizione il puntatore a fine lista ha consentito di aggiungere un nuovo elemento alla fine della lista senza dover prima scandire tutta la lista.

/

ITIS-LS F.Giordani Caserta

Anno scolastico 2014/2015

Classe 4^ sez.D spec. Informatica

Data: 13/01/2015

Numero es:2

Versione:1.0

Programmatore/i: Sebastiano Fusco

Sistema Operativo:Windows XP

Compilatore/Interprete:Dev-C++ 4.9.9.2

Obiettivo didattico: L'alunno e' in grado di usare le liste L.I.F.O. (Pila)

Obiettivo del programma: Pila di caratteri (Lista L.I.F.O.)

TABELLA DELLE VARIABILI				
IDENTIF.	*I/O/LAVORO*	*DESC.*	*TIPO*	*DIMENSIONE*
**p*	*I/O*		*STRUCT NODO*	
**testa*	*LAVORO*		*STRUCT NODO*	
scelta	*LAVORO*		*INTERO*	*4 BYTE*

**/*

```
#include<iostream>
#include<cstdlib>
using namespace std;
struct nodo
 {
  char carattere;
  struct nodo *next;
 };

int main()
{
 nodo *testa;
 nodo *p;
 testa=NULL;
 int scelta;
 do
 {
  cout<<"1)Inserimento carattere"<<endl;
  cout<<"2)Stampa caratteri"<<endl;
  cout<<"Fai una scelta: (0 per terminare) ";
```

```
cin>>scelta;
switch(scelta)
{
  case 1:
    p=new(nodo);
    cout<<"Inserire il carattere: ";
    cin>>p->carattere;
    p->next=testa;
    testa=p;
    cout<<endl;
  break;
  case 2:
    if(testa==NULL)
      cout<<"Non ci sono autovetture disponibili! Attendere!"<<endl;
    else
    {
      cout<<"I caratteri sono: "<<endl;
      for(p=testa;p!=NULL;p=p->next)
        cout<<p->carattere<<" ";
    }
    cout<<endl;
```

```
    break;
   }
  }
  while(scelta != 0);
  system("pause");
}
```

```
 Versione 1.1
#include<iostream>
#include<cstdlib>
using namespace std;
struct nodo
 {
  char carattere;
  struct nodo *next;
 };
int main()
{
  nodo *testa;
  nodo *p;
  testa=NULL;
  int scelta;
 do
 {
  cout<<"1)Push (inserimento)"<<endl;
  cout<<"2)Pop (estrazione)"<<endl;
  cout<<"3)Stampa lista dei caratteri"<<endl;
  cout<<"Fai una scelta: (0 per terminare) ";
  cin>>scelta;
  switch(scelta)
 {
   case 1:
     p=new(nodo);
    cout<<"Inserire il carattere: ";
    cin>>p->carattere;
    p->next=testa;
    testa=p;
    cout<<endl;
```

```
  break;
  case 2:
    p=testa;
   cout<<p->carattere<<endl;
   testa=p->next;
 break;
  case 3:
    if(testa==NULL)
   cout<<"lista vuota"<<endl;
   else
    {
      cout<<"I caratteri sono: "<<endl;
     //for(p=testa;p!=NULL;p=p->next)
    p=testa;
    while(p!=NULL)
    {
      cout<<p->carattere<<" ";
      p=p->next;
    }
   }
   cout<<endl;
  break;
 }
 }
 while(scelta != 0);
 system("pause");
}
```

/

ITIS-LS F.Giordani Caserta

Anno scolastico 2014/2015

Classe 4^ sez.D spec. Informatica

Data: 15/01/2015

Numero es:3

Versione:1.0

Programmatore/i: Sebastiano Fusco

Sistema Operativo:Windows XP

Compilatore/Interprete:Dev-C++ 4.9.9.2

Obiettivo didattico: L'alunno e' in grado di usare le liste F.I.F.O. (Coda)

Obiettivo del programma: Coda di pazienti (Lista F.I.F.O.)

TABELLA DELLE VARIABILI				
IDENTIF.	*I/O/LAVORO*	*DESC.*	*TIPO*	*DIMENSIONE*
**p*	*I/O*		*STRUCT PAZIENTE*	
**testa*	*LAVORO*		*STRUCT PAZIENTE*	
**coda*	*LAVORO*		*STRUCT PAZIENTE*	
scelta	*LAVORO*		*INTERO*	*4 BYTE*
num	*LAVORO*		*INTERO*	*4 BYTE*

**/*

```
#include <iostream>
#include <string>
using namespace std;
struct paziente
```

```
 {
  string nome;
  int num;
  struct paziente *next;
 };

int main()
{
 paziente *testa,*coda;
 int scelta;
 paziente *p;
 int num;
 testa=NULL;
 coda=NULL;
 num=1;
 do
 {
  cout<<"1. Inserisci paziente"<<endl;
  cout<<"2. Estrai paziente"<<endl;
  cout<<"3. Stampa elenco pazienti in attesa"<<endl;
  cout<<"0. Esci"<<endl;
```

```
cout<<"Fai una scelta: "<<endl;
cin>>scelta;
switch(scelta)
{
 case 1:
  p=new(paziente);
  cout<<"Inserire nome paziente: ";
  cin>>p->nome;
  p->num=num++;
  cout<<"Il paziente "<<p->nome<<" ha il numero "<<p->num<<endl;
  p->next=NULL;
  if(testa==NULL)
   testa=coda=p;
  else
  {
   coda->next=p;
   coda=p;
  }
 break;
 case 2:
  if(testa==NULL)
```

```
      cout<<"Non ci sono pazienti in coda"<<endl;
    else
    {
      p=testa;
      testa=p->next;
      cout<<"Il prossimo da visitare e' "<<p->nome<<" e ha il numero
"<<p->num<<endl;
      delete p;
    }
  break;
  case 3:
    if (testa==NULL)
      cout<<"Non ci sono pazienti in coda"<<endl;
    else
    {
      cout<<"Elenco pazienti in attesa: "<<endl;
      cout<<"NOME\tNUMERO"<<endl;
      for (p=testa;p!=NULL;p=p->next)
        cout<<p->nome<<"\t"<<p->num<<endl;
    }
  break;
```

```
    }
  }
  while(scelta!=0);
  system("pause");
}
```

```
/*
ITIS-LS F.Giordani Caserta
Anno scolastico 2014/2015
Classe 4^ sez.D spec. Informatica
Data: 12/01/2015
Numero es:4
Versione:1.0
Programmatore/i: Sebastiano Fusco
Sistema Operativo:Windows XP
Compilatore/Interprete:Dev-C++ 4.9.9.2
Obiettivo didattico: L'alunno e' in grado di usare le liste L.I.F.O. (Pila)
Obiettivo del programma: Pila di autovetture (Lista L.I.F.O.)
```

TABELLA DELLE VARIABILI

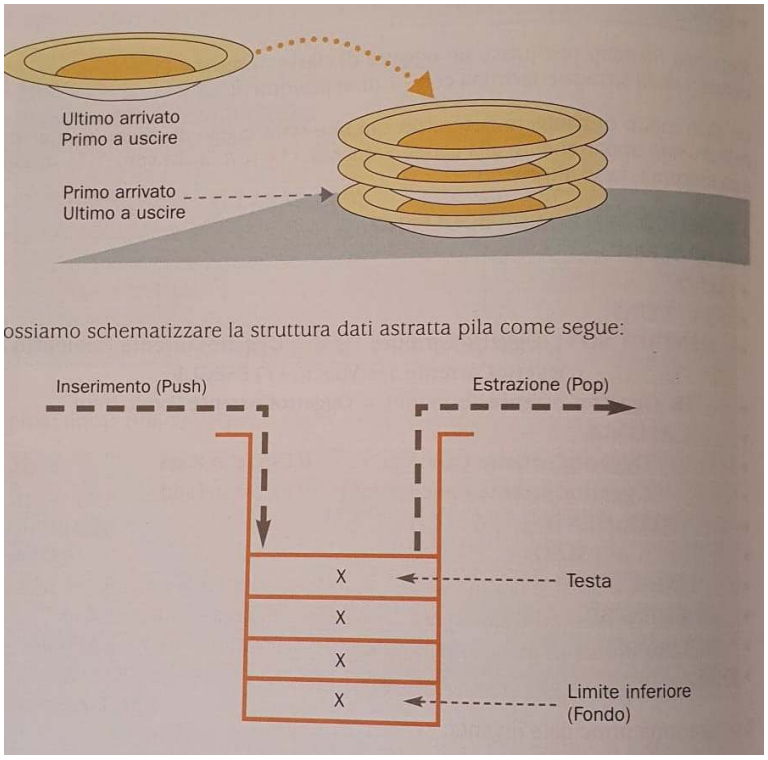
IDENTIF.	I/O/LAVORO	DESC.	TIPO	DIMENSIONE
*p	I/O		STRUCT VETTURA	
*testa	LAVORO		STRUCT VETTURA	
scelta	LAVORO		INTERO	4 BYTE

```
*/
#include<iostream>
#include<cstdlib>
#include<string>
using namespace std;
struct vettura
 {
   string targa;
   int cil;
   struct vettura *next;
 };
int main()
{
 vettura *testa;
 int scelta;
```

```
vettura *p;
testa=NULL;
do
{
 cout<<"1. Inscerisci auto (push)"<<endl;
 cout<<"2. Elimina auto (pop)"<<endl;
 cout<<"3. Stampa elenco auto presenti nel garage"<<endl;
 cout<<"0. Esci"<<endl;
 cout<<"Fai una scelta: ";
 cin>>scelta;
 switch(scelta)
 {
  case 1:
   p=new(vettura);
   cout<<"Inserire il numero di targa: ";
   cin>>p->targa;
   cout<<"Inserire la cilindrata: ";
   cin>>p->cil;
   p->next=testa;
   testa=p;
   cout<<endl;
```

```
    break;
    case 2:
     if(testa==NULL)
       cout<<"Non ci sono autovetture disponibili! Attendere!"<<endl;
     else
     {
       p=testa;
       testa=p->next;
       cout<<"La vettura assegnata ha targa "<<p->targa<<" e
cilindrata"<<p->cil<<endl;
       delete p;
     }
     cout<<endl;
    break;
    case 3:
     if(testa==NULL)
       cout<<"Non ci sono autovetture disponibili! Attendere!"<<endl;
     else
     {
       cout<<"Elenco autovetture disponibili: "<<endl;
       cout<<"TARGA\tCILINDRATA "<<endl;
       for(p=testa;p!=NULL;p=p->next)
         cout<<p->targa<<"\t"<<p->cil<<endl;
     }
     cout<<endl;
```

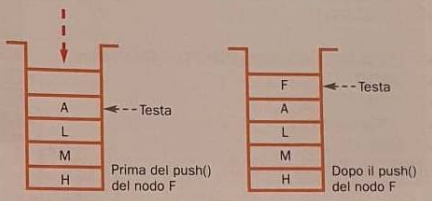
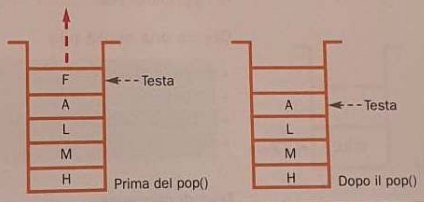
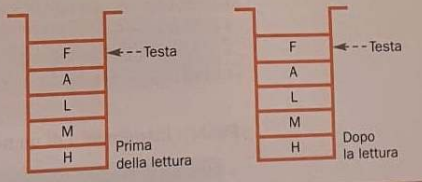
```
    break;
   }
  }
  while(scelta != 0);
  system("pause");
}
```



ossiamo schematizzare la struttura dati astratta pila come segue:

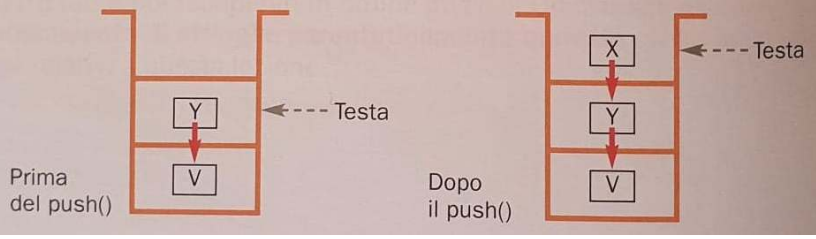
Operazioni sulla pila

Le operazioni possibili su una pila di nodi sono riassunte nella tabella che segue. Come puoi notare, le operazioni di inserimento e di prelievo da una pila prendono il nome di **Push** (inserimento) e **Pop** (prelievo).

Operazione	Descrizione	Situazione nella Sequenza prima e dopo l'operazione
Crea	Crea una nuova pila	
Testa pila vuota	Verifica se la pila è vuota	
Lunghezza	Restituisce il numero di nodi presenti nella pila	
Push: inserimento di un nuovo nodo in testa	Il nuovo nodo viene inserito nella prima posizione della pila	
Pop: estrazione di un nodo dalla testa della pila	Il nodo viene rimosso dalla testa della pila	
Leggi in testa	Legge l'elemento in testa alla pila senza rimuoverlo	
Ricerca di un nodo	La ricerca di un nodo avviene in modo sequenziale. Ciò significa che, per cercare un determinato nodo, occorre raggiungere quest'ultimo passando per tutti i nodi precedenti effettuando tanti pop() (cioè estraendo i nodi). Per lasciare la pila nella stessa situazione iniziale occorre utilizzare una pila di appoggio.	

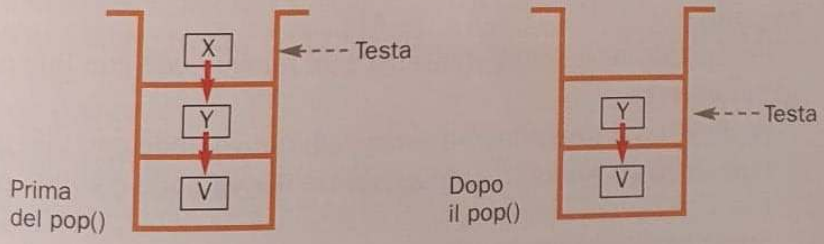
Push(): inserimento di un nodo

```
PUBBLICO Push(N: Nodo)
INIZIO
   N.Next ← OggettoCorrente.Testa      // Inserisce il nodo all'inizio della pila
   OggettoCorrente.Testa ← N           // Cambia la testa alla pila
FINE
```



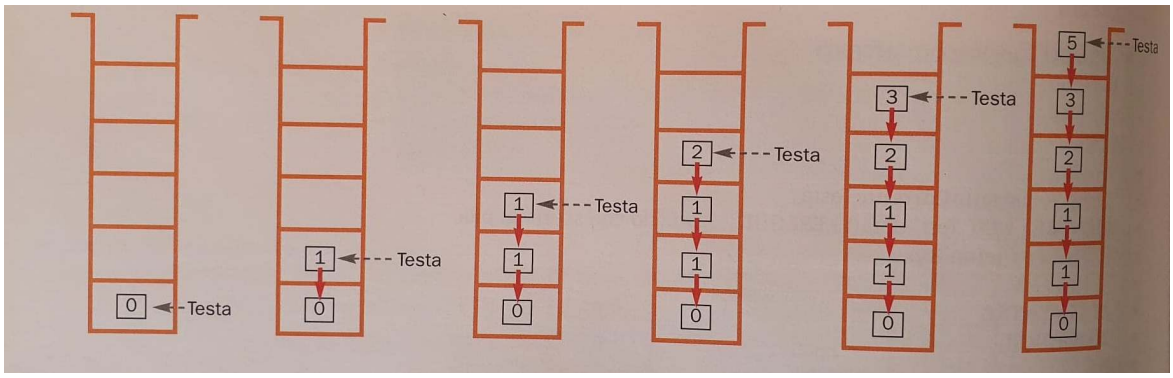
Pop(): estrazione di un nodo

```
PUBBLICO Pop(): Nodo
INIZIO
N: Nodo
   SE TestVuota()                      // Se la pila non è vuota
      ALLORA
         OggettoCorrente.Underflow()   // Chiama Underflow(), metodo privato
                                       // non visibile all'esterno
      ALTRIMENTI
         N ← OggettoCorrente.Testa
         OggettoCorrente.Testa ← OggettoCorrente.Testa.Next      // Cambia
                                       // la testa alla pila
         N.Next = NULL
   FINESE
   RITORNO(N)
FINE
```



Underflow

```
PRIVATO Underflow()
INIZIO
   SCRIVI("Pila vuota")          // Per semplicità (questo metodo deve gestire
                                 // la situazione di pila vuota
                                 // e potrebbe generare un'eccezione da programma
FINE
```




```
/*

ITIS-LS F.Giordani Caserta

Anno scolastico: 2020/2021

Classe 4° sez.B spec.Informatica

Data: 21/02/2020

Numero Esercizio: 5

Programmatore/i: Grauso Gianluca

Sistema Operativo: Windows 10

Compilatore/Interprete: Codeblocks 17.12

Obiettivo didattico: L'alunno è in grado di utilizzare I puntatori e le strutture concatenate
Obiettivo del programma: Eseguire le operazioni di aggiornamento relative ad una struttura concatenata con gestione Pila (L.I.F.O.)

*/

#include <iostream>

#include <stdlib.h>

using namespace std;
```

```
struct nodo{
	char inf;
	nodo* punt;
};

int Menu();
void StampaLista(nodo* testaPar);
int ricerca(nodo* testaPar, char kPar);
void Push(nodo* &testaPar, char carPar);
char Pop(nodo* &testaPar);
int InserisciDopo(nodo* testaPar, char dopo, char nuovo);
void ScambiaCar(nodo* testaPar,char car1, char car2);
int ModificaCar(nodo* testaPar, char oldCar, char newCar);
void EliminaCar(nodo* &testaPar, char carPar);

int main(){
	int scelta;
	nodo* testa = NULL;
	char car,car2;

	do{
```

```
scelta = Menu();
cout << "----------------------" << endl;
switch (scelta) {
case 1: {
        cout << "car da inserire = ";
        cin >> car;
        Push(testa, car);
        break;
}
case 2: {
        if (testa == NULL)cout << "La lista e' vuota" << endl;
        else cout << Pop(testa) << endl;
        break;
}
case 3: {
        StampaLista(testa);
        break;
}
case 4: {
        if (testa == NULL) cout << "La lista e' vuota" << endl;
        else {
```

```
            cout << "car da cercare = ";
            cin >> car;
            int pos = ricerca(testa, car);
            if (pos == -1) cout << "car non trovato" << endl;
            else cout << pos << endl;
        }
        break;
}
case 5: {
        cout << "car da cercare = ";
        cin >> car;
        cout << "car da inserire dopo = ";
        cin >> car2;
        int risultato = InserisciDopo(testa, car, car2);
        if(risultato == -1) cout<<"La lista e' vuota"<<endl;
        else if(risultato == 0) cout<<"car non trovato"<<endl;
        else cout<<"Inserimento avvenuto con successo"<<endl;
        break;
}
case 6: {
        cout << "primo car = ";
```

```
            cin >> car;
            cout << "secondo car = ";
            cin >> car2;
            ScambiaCar(testa,car,car2);
            break;
        }
        case 7: {
            cout << "car da modificare = ";
            cin >> car;
            cout << "car da inserire = ";
            cin >> car2;
            int risultato = ModificaCar(testa,car,car2);
if(risultato == -1) cout<<"La lista e' vuota"<<endl;
            else if(risultato == 0) cout<<"car non trovato"<<endl;
            else cout<<"Modifica avvenuta con successo"<<endl;
            break;
        }
        case 8: {
            cout << "car da eliminare = ";
            cin >> car;
            EliminaCar(testa, car);
```

```
            break;
        }
        case 9: {
            cout << "Programma Terminato" << endl;
            break;
        }
        default: {
            cout << "Opzione non valida" << endl;
            break;
        }
    }
    cout << "----------------------" << endl;
    system("PAUSE && cls");
    }while(scelta != 9);
    return 0;
}

int Menu(){
    int scelta;
    cout << "----------------------" << endl;
    cout << "1-Push" << endl;
```

```
	cout << "2-Pop" << endl;
	cout << "3-Stampa Lista" << endl;
	cout << "4-Ricerca" << endl;
	cout << "5-Inserisci dopo" << endl;
	cout << "6-Scambia" << endl;
	cout << "7-Modifica" << endl;
	cout << "8-Elimina" << endl;
	cout << "9-Esci" << endl;
	cout << "= ";
	cin >> scelta;
	return scelta;
}

void StampaLista(nodo* testaPar) {
	if (testaPar == NULL) cout << "La lista e' vuota" << endl;
	else{
		do{
			cout << testaPar->inf << endl;
			testaPar = testaPar->punt;
		}while (testaPar != NULL);
	}
```

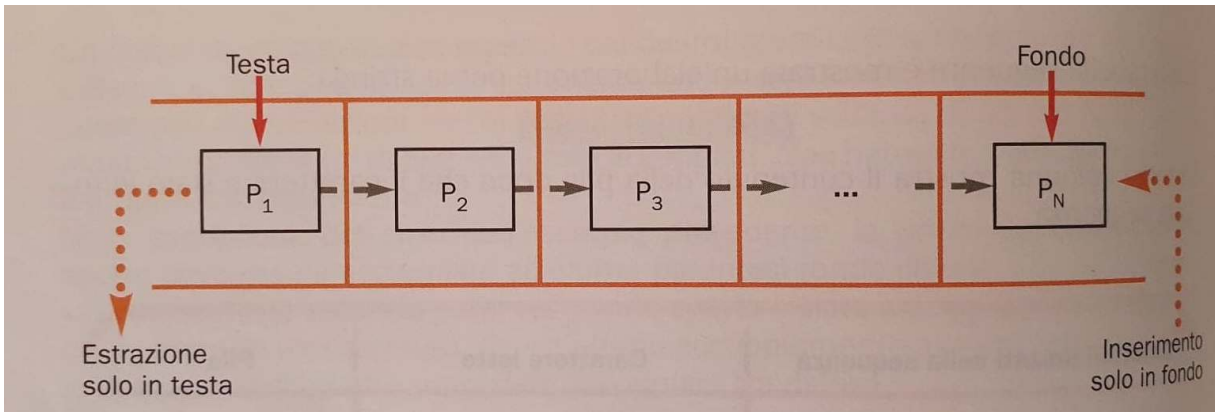
```
}

int ricerca(nodo* testaPar, char kPar) {
	int trovato = -1;
	if (testaPar != NULL) {
		int i = 1;
		do{
			if (testaPar->inf == kPar) trovato = i;
			else {
				testaPar = testaPar->punt;
				i++;
			}
		} while(trovato == -1 && testaPar != NULL);
	}
	return trovato;
}

void Push(nodo* &testaPar, char carPar) {
	nodo* newNodo = new(nodo);
	newNodo->inf = carPar;
	newNodo->punt = testaPar;
```

```
        testaPar = newNodo;
}

char Pop(nodo* &testaPar) {
        char tmpInf = testaPar->inf;
        testaPar = testaPar->punt;
        return tmpInf;
}

int InserisciDopo(nodo* testaPar, char dopo, char nuovo) {
   int returnCode=1;
        if (testaPar == NULL) returnCode=-1;
        else{
                int pos = ricerca(testaPar, dopo);
                if(pos == -1) returnCode=0;
                else {
                        for (int i = 1; i < pos; i++) testaPar = testaPar->punt;
                        nodo* pTMP = new(nodo);
                        pTMP->inf = nuovo;
                        pTMP->punt = testaPar->punt;
                        testaPar->punt = pTMP;
```

```
            }
        }
        return returnCode;
}
void ScambiaCar(nodo* testaPar,char car1,char car2) {
        if (testaPar == NULL) cout << "La lista e' vuota" << endl;
        else{
                int pos1 = ricerca(testaPar, car1);
                if (pos1 == -1) cout << "primo car non trovato" << endl;
                else {
                        int pos2 = ricerca(testaPar, car2);
                        if (pos2 == -1) cout << "secondo car non trovato" << endl;
                        else{
                                nodo* p1 = testaPar;
                                nodo* p2 = testaPar;
                                for (int i = 1; i < pos1; i++) p1 = p1->punt;
                                for (int i = 1; i < pos2; i++) p2 = p2->punt;
                                char tmpCar = p1->inf;
                                p1->inf = p2->inf;
                                p2->inf = tmpCar;
                                cout << "Scambio avvenuto con successo" << endl;
```

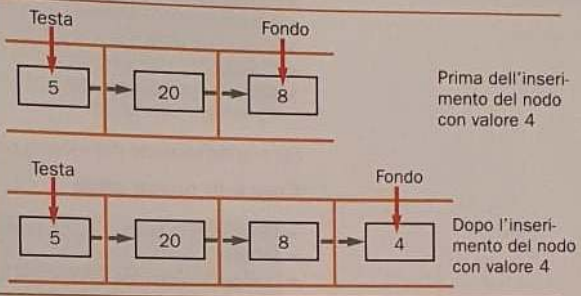
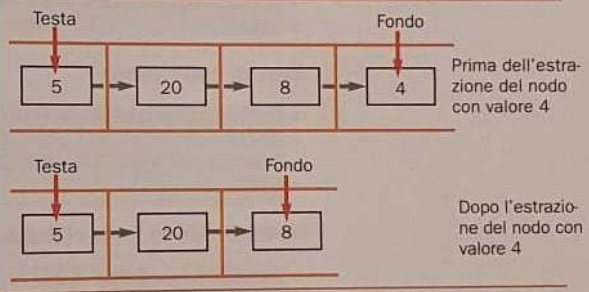
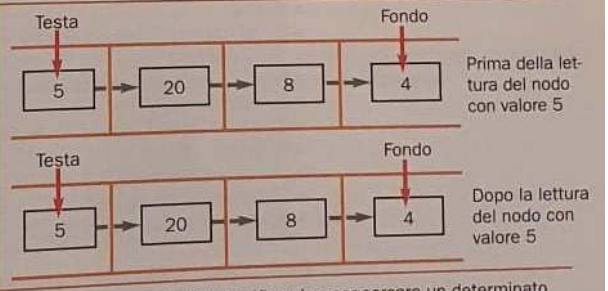
```
                }
            }
        }
}
int ModificaCar(nodo* testaPar, char oldCar, char newCar) {
        int returnCode=1;
        if (testaPar == NULL) returnCode=-1;
        else {
            int pos = ricerca(testaPar, oldCar);
            if (pos == -1) returnCode=0;
            else{
                for(int i = 1; i < pos; i++) testaPar = testaPar->punt;
                testaPar->inf = newCar;
                cout << "Modifica avvenuta con successo" << endl;
            }
        }
        return returnCode;
}

void EliminaCar(nodo* &testaPar, char carPar) {
   nodo* pTMP;
   pTMP = testaPar;
```

```
        if (pTMP == NULL) cout << "La lista e' vuota" << endl;
        else {
                int pos = ricerca(pTMP, carPar);
                if (pos == -1) cout << "car non trovato" << endl;
                else{
                        if(pos == 1){
                           Pop(testaPar);
                           cout << "Cancellazione avvenuta con successo" << endl;
                        }
                        else {
                                for (int i = 1; i < pos - 1; i++) pTMP = pTMP->punt;
                                if (pTMP->punt != NULL) {
                                        pTMP->punt = pTMP->punt->punt;
                                }
          cout << "Cancellazione avvenuta con successo" << endl;
                        }
                }
        }
}
```



Operazioni sulla coda

Le operazioni possibili su una coda di nodi sono riassunte nella tabella che segue.

Operazione	Descrizione	Situazione nella coda prima e dopo l'operazione
Crea	Crea una nuova coda	
Testa pila vuota	Restituisce vero se la coda è vuota, falso altrimenti	
Lunghezza	Restituisce il numero di nodi presenti nella coda	
Inserimento di un nuovo nodo in fondo (***enqueue***)	Il nuovo nodo viene inserito in fondo alla coda, in ultima posizione	
Estrazione di un nodo dalla testa (***dequeue***)	Viene rimosso il primo nodo inserito nella struttura	
Leggi in testa	Legge l'elemento in testa alla coda senza rimuoverlo	
Ricerca di un nodo	La ricerca di un nodo avviene in modo sequenziale. Ciò significa che, per cercare un determinato nodo, occorre raggiungere quest'ultimo passando per tutti i nodi precedenti.	

```
/*
ITIS-LS F.Giordani Caserta
Anno scolastico: 2020/2021
Classe 4° sez.B spec.Informatica
Data: 2/03/2020
Numero Esercizio: 6
Programmatore/i: Grauso Gianluca
Sistema Operativo: Windows 10
Compilatore/Interprete: Codeblocks 17.12
Obiettivo didattico: L'alunno è in grado di utilizzare I puntatori e le strutture concatenate
Obiettivo del programma: Eseguire le operazioni di aggiornamento relative ad una struttura concatenata con gestione Coda (F.I.F.O)
*/
#include <iostream>
#include <stdlib.h>
using namespace std;

struct nodo {
        char car;
        nodo* punt;
};
```

```
int Menu();
void Push(nodo* &testaPar,nodo* &codaPar,char carPar);
char Pop(nodo* &testaPar);
void Stampacoda(nodo* testaPar);
int ricercacar(nodo* testaPar,char kPar);
int Modificacar(nodo* testaPar,char oldCar,char newCar);
int InserisciDopo(nodo* testaPar,nodo* &codaPar,char dopo,char nuovo);
void Scambiacar(nodo* testaPar,char car1,char car2);
int Eliminacar(nodo* &testaPar,nodo* &codaPar,char carPar);

int main() {
	nodo* testa = NULL, * coda = NULL;
	int scelta;
	char car,car2;

	do{
	   scelta = Menu();
		cout<<"--------------------"<<endl;
		switch(scelta){
		   case 1:{
```

```
            cout<<"Inserire car = ";
            cin>>car;
            Push(testa,coda,car);
            break;
}
case 2:{
   if(testa == NULL) cout<<"La coda e' vuota"<<endl;
   else cout<<Pop(testa)<<endl;
   break;
}
case 3:{
   Stampacoda(testa);
   break;
}
case 4:{
   if(testa == NULL) cout<<"La coda e' vuota"<<endl;
   else{
      cout<<"car da cercare = ";
      cin>>car;
      int risultato = ricercacar(testa,car);
      if(risultato == -1) cout<<"car non trovato"<<endl;
```

```
        else cout<<risultato<<endl;
    }
    break;
}
case 5:{
    cout<<"car da cercare = ";
    cin>>car;
    cout<<"car da inserire dopo = ";
    cin>>car2;
    int risultato = InserisciDopo(testa,coda,car,car2);
    if(risultato == 1) cout<<"La coda e' vuota"<<endl;
    else if(risultato == 2) cout<<"car non trovato"<<endl;
    else cout<<"Inserimento avvenuto con successo"<<endl;
    break;
}
case 6:{
    cout<<"primo car = ";
    cin>>car;
    cout<<"secondo car = ";
    cin>>car2;
    Scambiacar(testa,car,car2);
```

```
    break;
}
case 7:{
    cout<<"car da modificare = ";
    cin>>car;
    cout<<"car da inserire = ";
    cin>>car2;
    int risultato = Modificacar(testa,car,car2);
    if(risultato == 1) cout<<"La coda e' vuota"<<endl;
    else if(risultato == 2) cout<<"car non trovato"<<endl;
    else cout<<"Modifica avvenuta con successo"<<endl;
    break;
}
case 8:{
    cout<<"car da eliminare = ";
    cin>>car;
    int risultato = Eliminacar(testa,coda,car);
    if(risultato == 1) cout<<"La coda e' vuota"<<endl;
    else if(risultato == 2) cout<<"car non trovato"<<endl;
    else cout<<"Cancellazione avvenuta con successo"<<endl;
    break;
```

```
        }
        case 9:{
            cout<<"Programma Terminato"<<endl;
            break;
        }
        default:{
            cout<<"Scelta non valida"<<endl;
            break;
        }
      }
            cout<<"--------------------"<<endl;
            system("PAUSE && cls");
   }while(scelta != 9);
   return 0;
}

int Menu(){
        int scelta;
        cout << "1-Push." << endl;
        cout << "2-Pop." << endl;
        cout << "3-Stampa coda." << endl;
```

```
        cout << "4-Ricerca car." << endl;
        cout << "5-Inserisci dopo." << endl;
        cout << "6-Scambia." << endl;
        cout << "7-Modifica car." << endl;
        cout << "8-Elimina." << endl;
        cout << "9-Esci." << endl;
        cout << "= ";
        cin >> scelta;
        return scelta;
}

void Push(nodo* &testaPar,nodo* &codaPar,char carPar){
   nodo* pTMP = new(nodo);
        pTMP->car = carPar;
        pTMP->punt = NULL;
        if(testaPar == NULL) testaPar = pTMP;
        else codaPar->punt = pTMP;
        codaPar = pTMP;
}

char Pop(nodo* &testaPar){
```

```
        char tmpCar = testaPar->car;
        testaPar = testaPar->punt;
        return tmpCar;
}

void Stampacoda(nodo* testaPar){
        if(testaPar == NULL) cout<<"La coda e' vuota"<<endl;
        else{
                do{
                        cout<<testaPar->car<<endl;
                        testaPar = testaPar->punt;
                }while(testaPar != NULL);
        }
}

int ricercacar(nodo* testaPar,char kPar){
        int trovato = -1;
        if(testaPar != NULL){
                int i = 1;
                do{
                        if(testaPar->car == kPar) trovato = i;
```

```
                else {
                        testaPar = testaPar->punt;
                        i++;
                }
        }while(trovato == -1 && testaPar != NULL);
    }
    return trovato;
}

int Modificacar(nodo* testaPar,char oldName,char newName){
    int codiceErrore = 0;
    if(testaPar == NULL) codiceErrore = 1;
    else{
        int pos = ricercacar(testaPar,oldName);
        if(pos == -1) codiceErrore = 2;
        else{
            for(int i = 1; i < pos; i++) testaPar = testaPar->punt;
            testaPar->car = newName;
        }
    }
    return codiceErrore;
```

```
}

int InserisciDopo(nodo* testaPar,nodo* &codaPar,char dopo,char nuovo){
    int codiceErrore = 0;
    if(testaPar == NULL) codiceErrore = 1;
    else{
        int pos = ricercacar(testaPar, dopo);
        if(pos == -1) codiceErrore = 2;
        else{
            for(int i = 1; i < pos; i++) testaPar = testaPar->punt;
            nodo* pTMP = new(nodo);
            pTMP->car = nuovo;
            pTMP->punt = testaPar->punt;
            testaPar->punt = pTMP;
            if(pTMP->punt == NULL) codaPar = pTMP;
        }
    }
    return codiceErrore;
}

void Scambiacar(nodo* testaPar,char car1,char car2) {
```

```
    if(testaPar == NULL) cout<<"La coda e' vuota"<<endl;
    else{
        int pos1 = ricercacar(testaPar,car1);
        if(pos1 == -1) cout<<"primo car non trovato"<<endl;
        else{
            int pos2 = ricercacar(testaPar,car2);
            if(pos2 == -1) cout<<"secondo car non trovato"<<endl;
            else{
                nodo* p1 = testaPar;
                nodo* p2 = testaPar;
                for(int i = 1; i < pos1; i++) p1 = p1->punt;
                for(int i = 1; i < pos2; i++) p2 = p2->punt;
                char tmpCar = p1->car;
                p1->car = p2->car;
                p2->car = tmpCar;
                cout<<"Scambio avvenuto con successo"<<endl;
            }
        }
    }
}
```

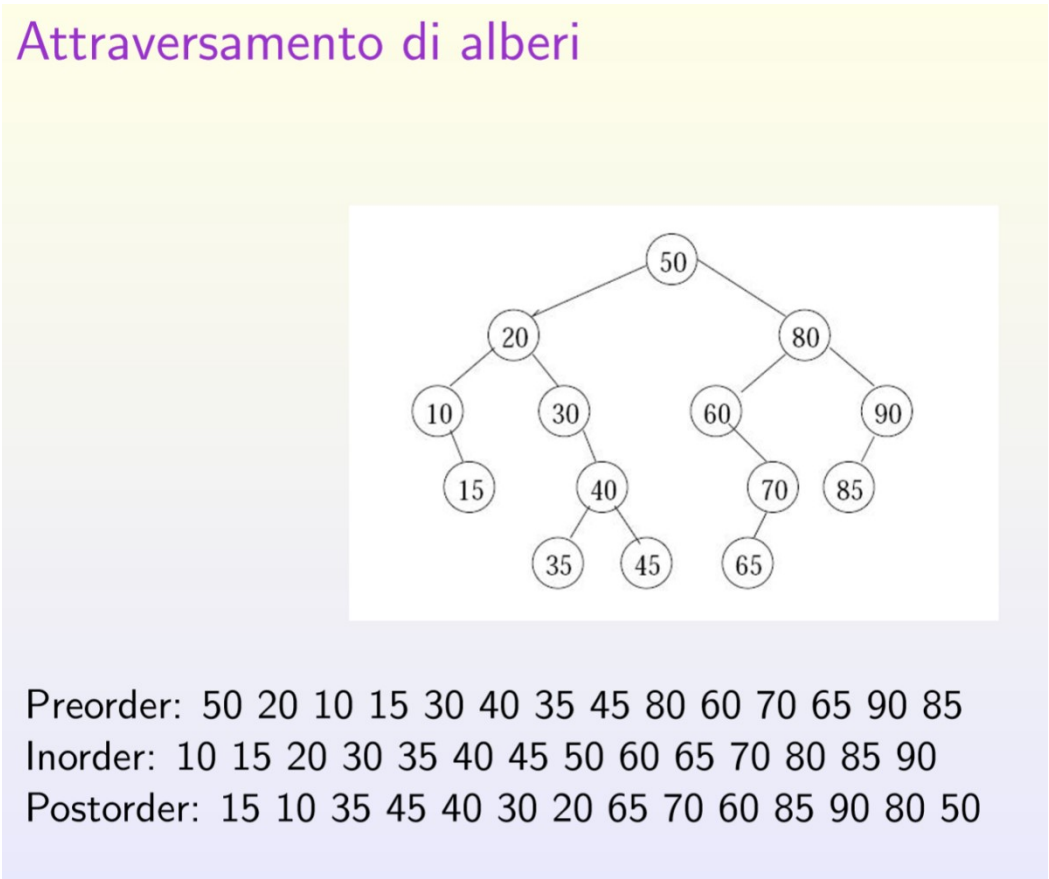
```
int Eliminacar(nodo* &testaPar,nodo* &codaPar,char carPar) {
	int codiceErrore = 0;
	nodo* pTMP = testaPar;
	if(pTMP == NULL) codiceErrore = 1;
	else{
		int pos = ricercacar(pTMP,carPar);
		if(pos == -1) codiceErrore = 2;
		else{
			if(pos == 1) Pop(testaPar);
			else{
				for(int i = 1; i < pos - 1; i++) pTMP = pTMP->punt;
				pTMP->punt = pTMP->punt->punt;
				if(pTMP->punt == NULL) codaPar = pTMP;
			}
		}
	}
	return codiceErrore;
}
```

Alberi

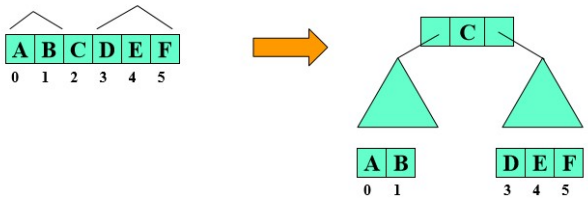
*Un **grafo** G é costituito da una coppia di insiemi(V,A) dove V é detto insieme dei nodi e A é detto insieme di archi ed é un sottinsieme di tutte le possibili coppie di nodi in V. Se le coppie di nodi sono ordinate, Il grafo é detto orientato,se non sono ordinate é detto non orientato.*
esempio
Grafo G con insieme di nodi V={a,b,c,d,e}, insieme di archi A={(a,b);(a,c);(b,c);(b,e);(c,d);(d,b)}
G grafo orientato:coppie ordinate e quindi,ad esempio,la coppia(a,b) é diversa dalla coppia(b,a)
G non orientato:coppie non ordinate e quindi,ad esempio,la coppia(a,b) e la coppia(b,a) sono equivalent tra loro.

*Un **albero** è un grafo orientato in cui da ogni nodo possono discendere più figli ma tale che ogni nodo, ad eccezione del nodo generatore (radice) abbia un solo padre. Un nodo è un elemento dell'albero.*
Un albero è una lista non lineare(un elemento può essere una lista)
Il nodo da cui discende un altro nodo è detto padre. I nodi discendenti da un nodo padre sono detti figli.

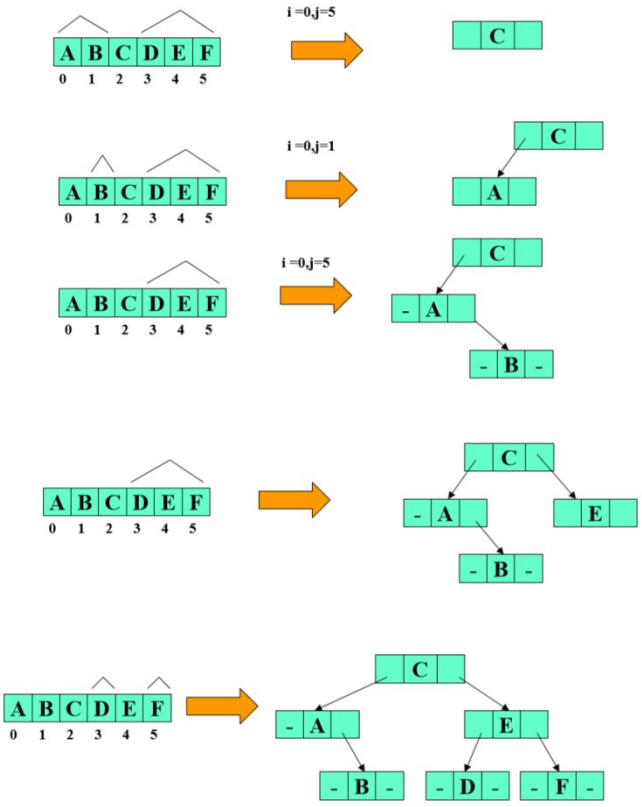
Per visitare un albero (cioè visitarne tutti i nodi, cioè gli elementi) esistono tre modi, secondo l'ordine di visita, tutti definiti ricorsivamente.
PREORDINE (Preorder) o ordine anticipato: R - S – D
POSTORDINE (Postorder) o ordine posticipato o polacco: S - D – R
INORDINE (Inorder) o ordine simmetrico: S - R – D
Dove R sta per la radice, D sta per sottoalbero destro della radice e S sta invece per sottoalbero sinistro della radice.



A partire da un vettore di interi possiamo costruire ricorsivamente un albero bilanciato nel numero di nodi selezionando a ogni chiamata una radice alla metà del vettore su cui si effettua la chiamata.



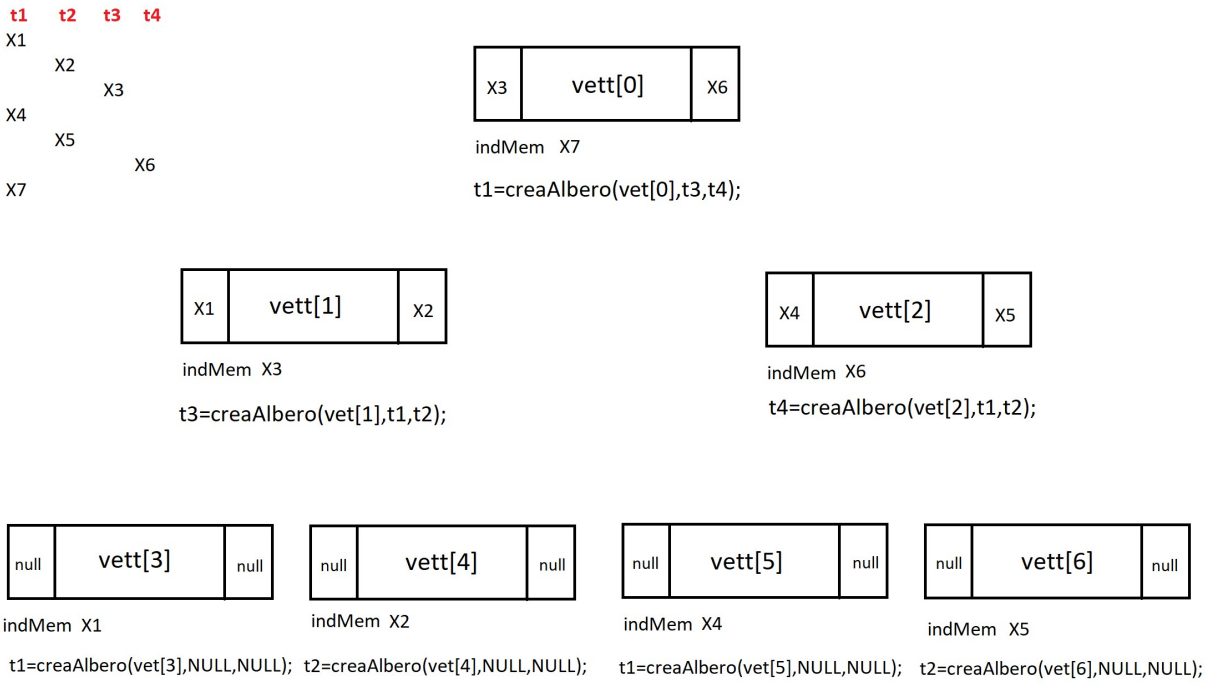

```
TreePtr AlbBinBil(int * vett, int i, int j)
/*costruisce un albero n-bilanciato a partire da un vettore  di indici i e j
* prec: vett!=NULL && i ≥0 && j≥0
*postc: t punta alla radice di un albero n-bilanciato che contiene vett[i],…,vett[j]*/
{int m;
TreePtr t;
if (i <= j)
  {m =  (i+j)/2;
    t = malloc(sizeof(Node));
    if (t)   t -> elem = vett[m];
    else
 /* malloc ha restituito un puntatore nullo */
fprintf(stderr,"%d non inserito, memoria non disponibile per  AlbBibBil.\n",vett[m]);
t->lPtr = AlbBinBil(vett,i,m-1);
t->rPtr = AlbBinBil(vett,m+1,j);
    return t;
  }
return NULL;
}
```



```
/*
ITIS-LS  Francesco Giordani Caserta
Anno scolastico :2019-2020
classe 4^B sez. Informatica
data:28/03/20
versione: 1.0
es. 7
programmatore: Di Palo Lorenzo
S.O.: windows 10
compilatore: Code-Blocks
Obiettivo didattico: introduzione agli Alberi binari
Obiettivo del  programma: Copiare gli elementi di un vettore in  un albero binario
 */
#include <iostream>
#include <stdlib.h>
using namespace std;
struct nodo
{
   int inf;
   nodo *puntNodoSx, *puntNodoDx;
};
typedef nodo *puntNodo;
puntNodo creaAlbero(int elementoRadice,  puntNodo sx, puntNodo dx)
{
 puntNodo  testaPar = new nodo;
 testaPar->puntNodoSx=sx;
 testaPar->puntNodoDx=dx;
 testaPar->inf=elementoRadice;
 return testaPar;
}
void stampa(puntNodo testaPar)
{
   if (testaPar)
  {
    cout<<testaPar->inf<<" ";
    stampa(testaPar->puntNodoSx);
```

```
    stampa(testaPar->puntNodoDx);
  }
}
void caricaVet(int dimlogPar,int vetPar[])
{
   for (int i=0;i<dimlogPar;i++)
     {
         cout<<"inserisci numero : ";
         cin>>vetPar[i];
      }
}
int main()
{
   nodo* t1,*t2,*t3,*t4;
   const int dimFis=7;
   int vet[dimFis], dimlog;
   dimlog=7;
   caricaVet(dimlog,vet);
   t1=creaAlbero(vet[3],NULL,NULL);
   t2=creaAlbero(vet[4],NULL,NULL);
   t3=creaAlbero(vet[1],t1,t2);
   t1=creaAlbero(vet[5],NULL,NULL);
   t2=creaAlbero(vet[6],NULL,NULL);
   t4=creaAlbero(vet[2],t1,t2);
   t1=creaAlbero(vet[0],t3,t4);
   stampa(t1);
}
```

Vengono creati i due nodi foglie della parte sinistra dell'albero (t1 , t2) collegati con t3, ottenendo un primo albero formato da radice, foglia Sx e Dx Ripetendo lo stesso procedimento per la parte destra dell'albero e poi collegando t3 (cioè la parte sinistra) e t4 (la parte destra) a t1, abbiamo creato l'albero in figura:



```
/*

  Nome progetto: Alberi binari
  Autore: Alessio Luffarelli
  Web: www.alessioluffarelli.it
  Data: Dicembre 2001

*/

#include <iostream>
#include <stdlib.h>

using namespace std;

struct nodo {
        int DATO; //albero di valori interi
        struct nodo *DX; //puntatore al sottoalbero destro
        struct nodo *SX; //puntatore al sottoalbero sinistro
        }NODO;

typedef struct nodo* tree; //dichiaro un nuovo tipo di dato: l'albero

bool Is_Empty(tree RADICE)
//restituisce TRUE se l'albero è vuoto. altrimenti restituisce FALSE
{
        return(RADICE==NULL); //NB: questo è un confronto (==) non un'assegnazione
}
```

```
tree Albero_Vuoto(void)
//costruisce un albero vuoto
{
        return(NULL);
}


int Valore_Etichetta(tree RADICE)
//restituisce il valore (o etichetta) del nodo
{
        if (Is_Empty(RADICE)) abort();
        else return(RADICE->DATO);
}

tree Sinistro(tree RADICE)
//restituisce il puntatore al sottoalbero sinistro
{
        if (Is_Empty(RADICE)) return(NULL);
        else return(RADICE->SX);
}

tree Destro(tree RADICE)
//restituisce il puntatore al sottoalbero destro
{
        if (Is_Empty(RADICE)) return(NULL);
```

```
        else return(RADICE->DX);
}

tree Costruisci_Albero(int ETICHETTA,tree S,tree D)
//costrisce un albero binario non ordinato
{
        tree RADICE;
        RADICE = (nodo *) malloc(sizeof(NODO)); //chiede l'indirizzo di una nuova cella di memoria
        RADICE->DATO = ETICHETTA;
        RADICE->SX = S;
        RADICE->DX = D;
        return (RADICE);
}

void Inorder(tree RADICE)
//stampa inorder dell'albero
{
        if (!(Is_Empty(RADICE))) {
                Inorder(Sinistro(RADICE));
                cout<<Valore_Etichetta(RADICE)<<"  ";
                Inorder(Destro(RADICE));
                }
}


void Preorder(tree RADICE)
//stampa preorder dell'albero
```

```
{
        if (!(Is_Empty(RADICE))) {
                cout<<Valore_Etichetta(RADICE)<<"  ";
                Preorder(Sinistro(RADICE));
                Preorder(Destro(RADICE));
                }
}


void Postorder(tree RADICE)
//stampa postorder dell'albero
{
        if (!(Is_Empty(RADICE))) {
                Postorder(Sinistro(RADICE));
                Postorder(Destro(RADICE));
                cout<<Valore_Etichetta(RADICE)<<"  ";
                }
}


int ContaNodi(tree RADICE)
//restituisce il numero dei nodi dell'albero
{
        if(Is_Empty(RADICE)) return(0);
        else return(1 + ContaNodi(Sinistro(RADICE)) + ContaNodi(Destro(RADICE)));
}
```

```
int ContaFoglie(tree RADICE)
//restituisce il numero delle foglie dell'albero
{
	if(Is_Empty(RADICE))
		return(0);
	else {
		if ((Sinistro(RADICE)==NULL) && (Destro(RADICE)==NULL))
			return(1);
		else return( ContaFoglie(Sinistro(RADICE)) + ContaFoglie(Destro(RADICE)) );
	}
}

bool Perf_Bil(tree RADICE)
//Restituisce TRUE se l'albero è perfettamente bilanciato
{
	if (Is_Empty(RADICE)) return(true);	//Oppure ERROR, secondo la definizione
	else {
		if ((Sinistro(RADICE)==NULL) && (Destro(RADICE)==NULL))	//Praticamente: se il nodo è una foglia..
			return(true);
		else {
			if ((Sinistro(RADICE)!=NULL) && (Destro(RADICE)!=NULL))	//Se il nodo ha tutti e due i figli..
				return( Perf_Bil(Sinistro(RADICE)) && Perf_Bil(Destro(RADICE)) );
			else return(false);
		}
	}
}
```

```
bool Ricerca(tree RADICE,int X)
//ricerca il valore X nell'albero puntato da RADICE
{
        if (Is_Empty(RADICE))
                return(false);
        else {
                if (X==Valore_Etichetta(RADICE))
                        return(true);
                else
                        return(Ricerca(Sinistro(RADICE),X) || Ricerca(Destro(RADICE),X));
        }
}


int Altezza_Nodo(tree N)        //Restituisce l'altezza di un nodo (vedere definizione di altezza)
{
  int ALTD=0,ALTS=0;
        if (Is_Empty(N)) return(-1);
        else {
                ALTS=Altezza_Nodo(Sinistro(N));
                ALTD=Altezza_Nodo(Destro(N));
                if (ALTS>ALTD) return(ALTS+1);
                else return(ALTD+1);
        }
}
```

```
tree Ins_Ord(int E,tree RADICE)
//Costruisce un albero binario di ricerca (un albero "ordinato")
{
	if (Is_Empty(RADICE)) {
		RADICE=(tree)malloc(sizeof(NODO)); //chiede l'indirizzo di una cella di memoria libera
		RADICE->DATO=E;
		RADICE->SX=NULL;
		RADICE->DX=NULL;
		return RADICE;
	}
	else {
		if(E<Valore_Etichetta(RADICE)) {
			RADICE->SX=Ins_Ord(E,Sinistro(RADICE));
			return RADICE;
		}
		else {
			RADICE->DX=Ins_Ord(E,Destro(RADICE));
			return RADICE;
		}
	}
}


bool RicercaBinaria(int X,tree RADICE)
//Ricerca dicotomica (o binaria) per alberi binari di ricerca (ordinati)
{
```

```
        if (Is_Empty(RADICE)) return(false);
        else {
                if (X==Valore_Etichetta(RADICE)) return(true);
                else {
                        if (X < Valore_Etichetta(RADICE))
                                return( RicercaBinaria(X,Sinistro(RADICE)) );
                        else
                                return( RicercaBinaria(X,Destro(RADICE)) );
                }
        }
}


// ---------------------------------------------
// Inizio del Main
//
// Il programma che segue mostra come applicare le
// funzioni di cui sopra

int main(int argc, char *argv[])
{

        tree t1,t2;
   tree t3,t4;
        int X;

        t1=Costruisci_Albero(2,Albero_Vuoto(),Albero_Vuoto());
```

```
	t2=Costruisci_Albero(1,Albero_Vuoto(),Albero_Vuoto());
	t3=Costruisci_Albero(7,t1,t2);

	t1=Costruisci_Albero(4,Albero_Vuoto(),Albero_Vuoto());
	t2=Costruisci_Albero(9,Albero_Vuoto(),Albero_Vuoto());
	t4=Costruisci_Albero(3,t1,t2);

	t1=Costruisci_Albero(5,t3,t4);
/*
In t1 c'è ora l'albero (o meglio, il puntatore all'albero)
rappresentato qui sotto:

                         5(t1)
                    /           \

               7                      3

              /\                     /\

          2         1         4           9(t2)
	*/

	cout<<"\nStampa in Inorder:\n";
	Inorder(t1);
```

```
cout<<"\n\nStampa in Preorder:\n";
Preorder(t1);

cout<<"\n\nStampa in Postorder:\n";
Postorder(t1);

cout<<"\n\nLa radice dell 'albero è : "<<Valore_Etichetta(t1);

cout<<"\n\nIl numero totale dei nodi dell'albero è : "<<ContaNodi(t1);

cout<<"\n\nIl numero totale delle foglie dell'albero è : "<<ContaFoglie(t1);

cout<<"\n\nQuesto albero ha altezza "<<Altezza_Nodo(t1)<<" ("<<Altezza_Nodo(t1)+1<<" livelli)";

if (Perf_Bil(t1)) cout<<"\n\nQuesto albero è perfettamente bilanciato.";
else cout<<"\n\nQuesto albero non è perfettamente bilanciato.";

cout<<"\n\nInserire un valore da cercare nell'albero : ";
cin>>X;
if (Ricerca(t1,X)) cout<<"\n\tValore trovato!!";
else cout<<"\n\tValore non trovato!!";

t1=Albero_Vuoto();

cout<<"\n\n\n*****************************************************************";
cout<<"\n********** COSTRUZIONE DI UN ALBERO BINARIO DI RICERCA **********";
```

```
cout<<"\n**********************************************************************";
cout<<"\n\n\nInserire tutti gli elementi (per terminare inserire 0)\n";

do {
        cout<<"\tElemento : ";
        cin>>X;
        t1=Ins_Ord(X,t1);
} while (X!=0);

cout<<"\nStampa in Inorder:\n";
Inorder(t1);

cout<<"\n\nStampa in Preorder:\n";
Preorder(t1);

cout<<"\n\nStampa in Postorder:\n";
Postorder(t1);

cout<<"\n\nLa radice è : "<<Valore_Etichetta(t1);

cout<<"\n\nIl numero totale dei nodi dell'albero è : "<<ContaNodi(t1);

cout<<"\n\nIl numero totale delle foglie dell'albero è : "<<ContaFoglie(t1);

cout<<"\n\nQuesto albero ha altezza "<<Altezza_Nodo(t1)<<" ("<<Altezza_Nodo(t1)+1<<" livelli)";

if (Perf_Bil(t1)) cout<<"\n\nQuesto albero è perfettamente bilanciato.";
```

```
        else cout<<"\n\nQuesto albero NON è perfettamente bilanciato.";

        cout<<"\n\nInserire un elemento da cercare con la ricerca dicotomica : ";
        cin>>X;

        if (RicercaBinaria(X,t1)) cout<<"\n\tValore trovato!!";
        else cout<<"\n\tValore non trovato!!";

        cout<<"\n\n\n\n\n";

  system("PAUSE");
  return 0;
}
```