ELENCO FUNZIONI

ARRAY

- inserimento
- visualizzazione
- ordinamento bubble sort
- ordinamento ingenuo
- ordinamento qsort
- ordinamento quicksort
- inserimento ordinato
- modifica elemento ordinato
- ricerca elemento
- ricerca elemento ordinato
- ricerca minimo
- ricerca massimo

LISTE

- creazione
- inserimento in testa
- inserimento in coda
- inserimento dopo elemento
- inserimento in lista ordinata
- ordinamento lista
- cancellazione in testa
- cancellazione in coda
- cancellazione di un elemento
- visualizzazione
- ricerca di un elemento

ALBERI

- creazione
- inserimento in albero bilanciato
- visualizzazione inorder
- visualizzazione postorder
- visualizzazione preorder
- cancellazione di un nodo
- ricerca di un elemento
- calcolo della profondità
- inserimento ordinato

FILE

- creazione
- inserimento in coda
- lettura

ARRAY

PROTOTIPI

```
void Inserisci_Elemento(int, int []);
void Ricerca_Sequenziale(int[], int);
void Trova_Massimo(int[], int);
void Trova_Minimo(int[], int);
void Ordinamento_Ingenuo(int[], int);
```

FUNZIONI

```
void Inserisci_Elemento(int n, int a[]) {
        int p,num;
        cout<< "inserire posizione ed elemento: " << endl;
        cin>> p >> num;
        if (p>num) {
                cerr <<" Posizione Errata" << endl;
                return;
        }
        a[p]=num;
}

void Ricerca_Sequenziale(int a[], int n) {
        bool trovato=false;
        int k;
        cout << " Inserisci il valore da trovare : " << endl;
        cin >> k;
        i=0;
        while ((i<n) && (!trovato)) {
                if (a[i] == k) trovato=true;
                i++;
        }
        if(trovato) cout<<”Elemento trovato”;
        else cout<<”Elemento non trovato”;
}

void Trova_Massimo(int a[], int n) {
        int Max = a[0];
        for(int i=1; i<n ; i++)
                if (a[i]>Max) Max=a[i];
        cout << " Il massimo è: " << Max << endl;
}

void Trova_Minimo(int a[], int n) {
        int Min = a[0];
        for(int i=1; i<n ; i++)
                if (a[i]<Min) Min=a[i];
        cout << " Il minimo è: " << Min << endl;
}

void Ordinamento_Ingenuo(int a[], int n) {
  for (int i=0; i<n-1; i++)
    for (int j=i+1; j<n; j++)
```

```
    if (a[j] < a[i]) {
        int t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
}
```

ALBERI

STRUCT

```
struct nodo {
  char word[30];
  nodo *sin, *des;     // puntatori ai sottoalberi sinistro e destro
  };
```

PROTOTIPO

```
void visualizza_inorder(nodo*);
void visualizza_preorder(nodo*);
void visualizza_postorder(nodo*);
void inserisci_albero(nodo*&, char*);
void ricerca(nodo*, char*, bool&);
```

FUNZIONI

```
void visualizza_inorder(nodo* albero) {
 if (!albero) return;                 //se è =0 chiudo la chiamata
 visualizza_inorder(albero->sin);          //richiamo la procedura passo il ramo SX
 cout << albero->word << endl;      //questa è il contenuta della foglia dell'albero e stampo
 visualizza_inorder(albero->des);          //richiamo la procedura e passo il ramo di destra
}

void visualizza_preorder(nodo* albero) {
 if (!albero) return;                 //se è =0 chiudo la chiamata
 cout << albero->word << endl;      //questa è il contenuta della foglia dell'albero e stampo
 visualizza_preorder(albero->sin);           //richiamo la procedura passo il ramo SX
 visualizza_preorder(albero->des);          //richiamo la procedura e passo il ramo di destra
}

void visualizza_postorder(nodo* albero) {
 if (!albero) return;                 //se è =0 chiudo la chiamata
 visualizza_postorder(albero->sin);          //richiamo la procedura passo il ramo SX
 visualizza_postorder(albero->des);          //richiamo la procedura e passo il ramo di destra
 cout << albero->word << endl;      //questa è il contenuta della foglia dell'albero e stampo
}

void inserisci_albero(nodo*& albero, char* w) {
  if (!albero) { // albero vuoto
   albero = new nodo;
   albero->word = new char[strlen(w)+1];
   strcpy(albero->word,w);
   albero->sin = 0; // inizializzo i rami a 0
   albero->des = 0;
  }
```

```
  else {
    int cmp = strcmp(w,albero->word);
    if (cmp<0) inserisci_albero(albero->sin,w);
    else if (cmp>0) inserisci_albero(albero->des,w);
  }
}

void ricerca(nodo* albero, char* w, bool& trovato) {
  if (albero->word == w) {
    trovato=true;
    return;
  }
  ricerca(albero->sin,w,trovato);          //richiamo la procedura passo il ramo SX
  ricerca(albero->des,w,trovato);          //richiamo la procedura e passo il ramo di destra
}
```

LISTE

```
STRUCT
struct lista {
        char nome[30];
        lista* succ;
};

PROTOTIPI
void crea_lista(lista*&);
void inserisci_testa(lista*&);
void inserisci_coda(lista*&);
void inserisci_dopo_elemento(lista*,char[]);
void inserisci_lista_ordinata(lista*&);
void cancellazione_testa(lista*&);
void cancellazione_coda(lista*&);
void cancellazione(lista*&,char[]);
void visualizza(lista*);
void ricerca(lista*);

FUNZIONI
void crea_lista(lista*& testa) {
        lista* nuovo;
        char n[30];
        cout<<"Inserisci i nomi (invio per terminare l'inserimento): "<<endl;
        testa=NULL;
        while ((cin.peek()!='\n')&&(cin>>n)) {
                nuovo=new lista;
                strcpy(nuovo->nome,n);
                nuovo->succ=testa;
                testa=nuovo;
        }
}

void inserisci_testa(lista*& testa) {
        lista* nuovo;
```

```
        nuovo=new lista;
        cout<<"Inserisci il nome: ";
        cin>>nuovo->nome;
        if (nuovo->nome==NULL) return;
        nuovo->succ=testa;
        testa=nuovo;
}

void inserisci_coda(lista*& testa) {
        if(!testa->succ) {
                lista* nuovo;
                nuovo=new lista;
                cout<<"Inserisci il nome: ";
                cin>>nuovo->nome;
                nuovo->succ=NULL;
                testa->succ=nuovo;
                return;
        }
        inserisci_coda(testa->succ);
}

void inserisci_dopo_elemento(lista* testa, char n[]) {
        while (testa!=NULL) {
                if (!strcmp(testa->nome,n)) {
                        lista* q;
                        q=new lista;
                        cout<<"Inserisci il nome: ";
                        cin>>q->nome;
                        q->succ=testa->succ;
                        testa->succ=q;
                        return;
                }
                testa=testa->succ;
        }
        cerr<<"Elemento non trovato";
}

void inserisci_lista_ordinata(lista*& testa) {
        cout<<"Inserisci nome: ";
        lista* q;
        cin>>q->nome;
        if (strcmp(testa->nome,q->nome)>0) {
                q->succ=testa;
                testa=q;
                return;
        }
        lista* prec;
        lista* t;
        prec=testa;
        t=testa->succ;
        while (t!=NULL) {
                if ((strcmp(prec->nome,q->nome)<0)&&(strcmp(t->nome,q->nome)>0)) {
                        q->succ=t;
```

```
                        prec->succ=q;
                        return;
                }
                t=t->succ;
                prec=prec->succ;
        }
        prec->succ=q;
        q->succ=NULL;
}

void visualizza(lista* testa) {
        if(!testa) return;
        cout<<testa->nome<<endl;
        visualizza(testa->succ);
}

void cancellazione_testa(lista*& testa) {
        lista* p=testa;
        testa=testa->succ;
        delete p;
}

void cancellazione_coda(lista*& testa) {
        if(testa->succ) cancellazione_coda(testa->succ);
        delete testa->succ;
        testa->succ=NULL;
}

void cancellazione(lista*& testa, char n[]) {
        if(!strcmp(testa->nome,n)) cancellazione_testa(testa);
        lista* prec=testa;
        lista* t=testa->succ;
        while (t) {
                if (!strcmp(t->nome,n)) {
                        prec->succ=t->succ;
                        delete t;
                        return;
                }
                t=t->succ;
                prec=prec->succ;
        }
        cerr<<"Elemento non trovato";
}

void ricerca(lista* testa) {
        char n[30];
        cout<<"Inserisci cosa cercare: ";
        cin>>n;
        while (testa!=NULL) {
                if (!strcmp(testa->nome,n)) {
                        cout<<"ELEMENTO TROVATO";
                        return;
                }
```

```
            testa=testa->succ;
        }
        cerr<<"ELEMENTO NON TROVATO";
}
```

FILE

PROTOTIPI

```
void apertura_scrittura (void);
void apertura_lettura (void);
void apertura_append (void);
void apertura_visualizzazione (void);
void apertura_scrittura_binario (void);
```

FUNZIONI

```
void apertura_scrittura(void) {
  ofstream fout("belloebravofile.txt");    // apro il file
  fout << "Sono bello e bravo!" << endl;  // scrivo sul file
  fout.close();  // chiudo il file
}

void apertura_lettura(void) {
  ifstream fin("belloebravofile.txt");    // apro il file
  char testo[80];
  fin.getline(testo, 80);         // leggo una riga dal file
  fin.close();                  // chiudo il file
  cout << testo << endl;          // visualizzo la riga letta
}

void apertura_append(void) {
  ofstream fout("belloebravofile.txt", ios::app); // apro in modalita' append
  fout << "Sono molto bello...come Batman" << endl;
  fout.close();
}

void apertura_visualizzazione(void) {
  ifstream fin("belloebravofile.txt");    // apro il file
  char testo[80];
  while (fin.getline(testo,80))   // leggo una riga dal file
       cout << testo << endl;    // visualizzo la riga letta
  fin.close();   // chiudo il file
}

void apertura_scrittura_binario(void) {
  fstream fout(“file.dat”,ios::out | ios::binary);
  fout << "Sono bello e bravo!" << endl;  // scrivo sul file
  fout.close();  // chiudo il file
}
```