E-08: FILE IN C

FONDAMENTI DI INFORMATICA E LABORATORIO T-AB

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

UNIVERSITÀ DI BOLOGNA, A.A. 2008/2009

Paolo Torroni, Marco Montali

8 Maggio 2009

Esercizio 1.

Si progettino e implementino due funzioni:

- `intArr2file(...)` che salvi in un **file di testo** un **vettore di interi**,
- `file2IntArr(...)` che trasferisca in un **vettore di interi** il contenuto di un **file di testo**.

Nel file, si usi come separatore (tra un intero e l'altro) il punto e virgola e come terminatore (alla fine dell'array) il punto. Si possono usare allo scopo le funzioni `atoi` e `itoa`.

Esercizio 2.

Si estenda il tipo di dato astratto `List` con due funzioni:

- `serialize(...)` per scrittura su file di una istanza del tipo `List`,
- `deserialize(...)` per lettura da file.

Si implementino due versioni diverse per ciascuna funzione `serialize(...)`/`deserialize(...)`:

1. per il tipo `List` implementato mediante lista collegata, utilizzando i file di testo;
2. per il tipo `List` implementato mediante lista collegata, utilizzando i file binari.

Esercizio 3.

Per ciascuno dei 4 casi identificati al punto precedente, si ridefinisca il metodo `view` del tipo `List` utilizzando il metodo `serialize(...)`.

Risorse utili

Implementazione ad array

Dichiarazione del tipo `List`:

```
// costruttore di List (implem. array)
typedef int elem;
typedef struct {
    int N,M; // dimensione fisica (N) e logica (M)
    elem *V; // vettore di elementi
} List;
```

Costruttore del tipo di dato astratto `List`:

```
List *newList( int dim ) {
    elem *V; List *L;
    V=( elem* )malloc( dim*sizeof( elem ) );
    L=( List* )malloc( sizeof( List ) );
    L->V=V;
    L->N=dim;
    L->M=0;
    return L;
}
```

Interfaccia del tipo di dato astratto `List`:

```
// costruttore
// crea una lista nuova di dimens. fisica dim
List *newList( void );

// restituisce 1 se la lista e' vuota
int isEmpty( List* );

// restituisce la nuova dimens. della lista
List *insert( List*, elem );

// restituisce la nuova dimensione della lista
// oppure un valore negativo in caso di errore
List *insertAt( List*, elem, int );

// restituisce il primo elemento della lista
// precondizione: la lista non e' vuota
elem head( List* );

// rimuove l'istanza dalla memoria
void clear( List* );

// restituisce la dimensione logica
```

```
int size( List* );

// restituisce la dimensione fisica
int max_size( List* );

// concatena due liste
// se non c'e' abbastanza spazio nella prima
// per appendere la seconda, restituisce NULL
List *append( List*, List* );

// rimuove un elemento  dalla lista
// restituisce la nuova dimensione della lista
// ovvero -1 se elemento non trovato
List *delete( List*, elem );

// restituisce l'indice della posizione da cui e'
// stato trovato elem, ovvero -1 se non trovato
List *find( List*, elem );

// mostra tutti gli elementi della lista
void view( List*, char* );
```

Implementazione del metodo `view`:

```
void view( List *L, char *name ) {
    int i=0;
    printf( "\nList %s: df=%d, dl=%d\n[ ",
         name, max_size( L ), size( L ) );
    for( i=0; i<size( L ); i++ ) {
        viewElem( L->V[i] );
        if( i<size( L )-1 )
            printf( ", " );
    }
    puts( " ]" );
}
```

Implementazione a lista collegata

Dichiarazione del tipo `List`:

```
// costruttore di List (implem. lista collegata)
typedef int elem;
typedef struct List {
    elem value; // elemento della lista
    struct List *next; // puntatore al nodo succ.
} List;
```

Interfaccia del tipo di dato astratto `List`:

```
// costruttore: crea una lista nuova vuota
List *newList( void );

// restituisce 1 se la lista e' vuota
int isEmpty( List* );

// inserimento in testa
// restituisce il puntatore alla nuova lista
// oppure NULL in caso di errore
List *insert( List*, elem );

// restituisce il primo elemento della lista
// precondizione: la lista non e' vuota
elem head( List* );

// restituisce la lista composta da tutti gli
// elementi eccetto il primo di una lista data
// se la lista e' vuota restituisce NULL
List *tail( List* );

// rimuove l'stanza dalla memoria
void clear( List* );

// restituisce la dimensione (logica=fisica)
int size( List* );

// concatena due liste. Se non c'e' abbastanza
// spazio nella prima per appendere la seconda,
// restituisce NULL
List *append( List*, List* );

// restituisce un riferimento all'ultimo eleme.,
// ovvero NULL se la lista e' vuota
List *last( List* );

// rimuove un elemento dalla lista
// restituisce la nuova lista
// ovvero NULL se elemento non trovato
List *delete( List*, elem );

// restituisce il riferimento all'elem. trovato
// ovvero NULL se non trovato
List *find( List*, elem );

// mostra tutti gli elementi della lista
void view( List*, char* );

// restituisce un riferimento all'elem. succ.
// ovvero NULL se applicato all'ultimo elem.
List *next( List*, int );

// crea una copia di L
List *copy( List *L );
```

Implementazione del metodo `view`:

```
void view( List *L, char *name ) {
    printf( "\nList %s: d.log=%d\n[ ",
        name, size( L ) );
    viewL( L );
    puts( " ]" );
}

void viewL( List *L ) {
    if( L!=NULL ) {
        viewElem( L->value );
        if( L->next!=NULL )
            printf( ", " );
        viewL( L->next );
    }
}

// ipotesi: elem e' strutt. equiv. a int
void viewElem( elem e ) {
    printf( "%d", e );
}
```

Prototipi delle funzioni di uso comune

Prototipi delle funzioni di uso comune

Header	Interfaccia	Error
string.h	`size_t strlen( char * ); // size_t e' un tipo int`	
string.h	`char *strcpy( char *, const char * ); // copia`	
string.h	`char *strcat( char *, const char * ); // concatenazione`	
string.h	`int strcmp( const char *, const char * ); // confronto`	
stdlib.h	`void *malloc( size_t ); // restituisce un generico puntatore`	`NULL`
stdlib.h	`void free( void * );`	
stdio.h	`FILE *fopen( char* name, char *mode );`	`NULL`
stdio.h	`int fclose( FILE * );`	`EOF`
stdio.h	`int feof( FILE * ); // vero se file pointer su EOF`	
stdio.h	`int fseek( FILE *, long offs, int orig ); // 0 SEEK_SET, 1 SEEK_CUR, 2 SEEK_END`	
stdio.h	`void rewind( FILE * );`	
stdio.h	`long ftell( FILE * ); // byte a cui si trova il file pointer`	`-1`
stdio.h	`int fprintf( FILE *, char * [, ...] );`	
stdio.h	`int fscanf( FILE *, char * [, ...] );`	
stdio.h	`char *fgets( char *, int, FILE * ); // legge la riga intera fino a '\n'`	`NULL`
stdio.h	`int *fputs( char *, FILE * ); // scrive una stringa e aggiunge '\n'`	`EOF`
stdio.h	`int fread( void *vet, int size, int n, FILE *fp );`	
stdio.h	`int fwrite( void *vet, int size, int n, FILE *fp );`	

Funzioni `atoi` e `itoa`

```
/* <stdlib.h> */
int atoi ( const char * str );
char *itoa ( int value, char * str, int base );
```

atoi parses the C string **str** interpreting its content as an integral number, which is returned as an `int` **value**.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in **str** is not a valid integral number, or if no such sequence exists because either **str** is empty or it contains only whitespace characters, no conversion is performed.

Parameters (**atoi**)

- **str** C string beginning with the representation of an integral number.

Return Value (**atoi**) On success, the function returns the converted integral number as an `int` value. If no valid conversion could be performed, a **zero** value is returned. If the correct value is out of the range of representable values, `INT_MAX` or `INT_MIN` is returned.

itoa converts an integer **value** to a `NULL`-terminated string using the specified **base** and stores the result in the array given by **str** parameter.

If **base** is 10 and **value** is negative, the resulting string is preceded with a minus sign (-). With any other **base**, **value** is always considered unsigned.

Parameters (**itoa**)

- **value** Value to be converted to a string.
- **str** Array in memory where to store the resulting null-terminated string.
- **base** Numerical base used to represent the value as a string, between 2 and 36, where 10 means decimal base, 16 hexadecimal, 8 octal, and 2 binary.

Return Value (**itoa**) A pointer to the resulting null-terminated string, same as parameter str.