CREARE UNA LIBRERIA IN C

Prima di cominciare a vedere come fare una libreria, dobbiamo capire di cosa abbiamo bisogno...

- di un file *.h* che sarà l'header della libreria (ovvero il file che conterrà i prototipi delle funzioni della libreria e la definizione dei tipi che l'utente che utilizzerà la libreria potrà usare).
- di un file *.c* che sarà il body della libreria (ovvero conterrà il corpo delle funzioni e la definizione dei tipi base usati dalla funzioni).

Detto questo, passiamo ad alcune considerazioni prima di vedere la creazione di una vera e propria libreria... avendo un file .h in cui si definiscono i tipi e i prototipi, possiamo usare ciò per definire un tipo particolare che a variazione della sua definizione può essere **int, char, string** o qualsiasi altra cosa... vediamo come.
Per prima cosa, creiamo il nostro file *.h*:

```
//item.h
typedef struct item* Item;

int equals(Item x, Item y);
int compareTo(Item x, Item y);
Item readItem();
void printItem(Item i);
```

Questo file chiamato *item.h* come vedete contiene alcuni metodi per la lettura dei suoi dati, la stampa di essi e il loro confronto.

Il tipo **Item** è definito attraverso un **typedef** di un puntatore di una struttura chiamata **struct item**. Ma cosa sarà lo **struct item** per noi?
Grazie proprio a questa non “conoscenza” possiamo produrre più body per il file *.h*, così da avere più definizioni per il tipo di **struct item**, ma una sola per l'header, e quindi tutti coloro che vorranno usare l'oggetto Item non dovranno riscrivere più volte il loro codice cliente (ovvero il programma che userà la libreria), ma semplicemente decidere quale definizione di **Item** usare durante la compilazione.
Giusto per fare vedere due esempi, ecco qui due body per la libreria *item*:

Item-string.c	*Item-int.c*

`#include <stdlib.h>`
`#include <stdio.h>`
`#include <string.h>`
`#include "item.h"`
 	
`struct item`
`{`
`  char* value;`
`};`
 	
`int equals(Item x, Item y)`
`{`
`  return (strcmp(x->value, y-`
	`#include <stdlib.h>`
	`#include <stdio.h>`
	`#include "item.h"`
	
	`struct item`
	`{`
	`  int value;`
	`};`
	
	`int equals(Item x, Item y)`
	`{`
	`  int a,b;`

```
>value)==0);
}




int compareTo(Item x, Item y)
{
     return strcmp(x->value, y-
>value);

}









Item readItem()
{
     Item i =
malloc(sizeof(struct item));
     char string[512];
     char* ns;

     printf("Insert value to
read: ");
     gets(string);

ns=calloc(strlen(string),sizeof(
char));
     strcpy(ns, string);
     i->value=ns;

     return i;
}

void printItem(Item i)
{
     printf("Value of item:
%s \n", i->value);
}
```

```
     a=x->value;
     b=y->value;

     return a==b;
}

int compareTo(Item x, Item y)
{
     int a,b;

     a=x->value;
     b=y->value;

     if(a>b) return 1;
     else if(a==b) return 0;
     else return -1;

}

Item readItem()
{
     Item i =
malloc(sizeof(struct item));
     int n;

     printf("Insert value to
read: ");
     scanf("%d", &n);
     i->value=n;

     return i;
}




void printItem(Item i)
{
     printf("Value of item:
%d \n", i->value);
}
```

Come potete vedere dal corpo di queste due funzioni, entrambe hanno nel loro codice ciò:

```
#include "item.h"
```

Cosa significa ciò? Significa *"durante la compilazione, portati dentro le definizioni di tipi e funzioni dentro il file item.h".* L'uso dei doppi apici si riferisce che il file *item.h* si trova nella stessa directory del vostro file.

Ora che abbiamo sia i body che l'header possiamo creare la nostra libreria... ma come? Con il comando *ar*.
Assicurandovi che il *item.h* e i due *item-*.c* siano nella stessa directory date i seguenti comandi da terminale:

```
$ gcc -c item-int.c
$ ar ruv item-int.a item-int.o
$ gcc -c item-string.c
$ ar ruv item-string.a item-string.o
```

Ora dovreste avere nella vostra directory 2 archivi chiamati *item-int.a* e *item-string.a*.
Questi due archivi sono puro file binario che potrete distribuire insieme ai vostri file *.h* come librerie c compatibili col sistema architetturale in cui sono state compilate (*x86-32bit, x86-64bit, PowerPC, ARM,* etc..).
Ora vi chiederete.. e dunque? Qual'è l'utilità di ciò?
Per ora non si vede, ma andiamo avanti... vediamo ora di voler creare ora una libreria che gestisce uno stack di Item. Perché di **Item** e non di **int**? Perché quando l'utente che userà la libreria dello stack potrà autonomamente scegliere se usare la libreria che legge interi o stringhe, il tutto modificando solo il codice del proprio programma! (In questo modo le nostre librerie saranno autonome dal resto del programma, e quindi ri-utilizzabili in molteplici contesti).
Bado alle ciance, e vediamo per prima cosa il nostro file *.h*

```
//astrostack.h
#include "item.h"

typedef struct stack* AstroStack;

AstroStack initstack();
int        stackempty(AstroStack s);
void       push(AstroStack s, Item elem);
Item       pop(AstroStack s);
Item       top(AstroStack s);
int        size(AstroStack s);
```

In maniera identica come prima, definiamo i tipi e i prototipi delle funzioni che vogliamo che l'utente a cui distribuiamo tutto ciò possa utilizzare nel proprio programma.
Passiamo alla definizione del body...

```
//astrostack.c
#include <stdlib.h>
#include "astrostack.h"
#include "item.h"

typedef struct node {
     Item info;
     struct node* next;
} Node;

typedef Node* List;

struct stack {
```

```
        List contents;
        int size;
};

AstroStack initstack()
{
        AstroStack s;
        s = malloc(sizeof(struct stack));
        s->contents = NULL;
        s->size=0;

        return s;
}

int stackempty(AstroStack s)
{
        return s->contents == NULL;
}

void push(AstroStack s, Item elem)
{
        List l = malloc(sizeof(Node));
        l->info = elem;
        l->next = s-> contents;
        s->contents = l;
        s->size=(s->size)+1;
}

Item pop(AstroStack s)
{
        List old = s->contents;
        Item elem = old->info;
        s->contents = old->next;
        s->size=(s->size)-1;
        free(old);

        return elem;
}

Item top(AstroStack s)
{
        Item elem = (s->contents)->info;

        return elem;
}

int size(AstroStack s)
{
        return s->size;
}
```

Come potete vedere, la definizione del nostro stack non vincolata in alcun modo al tipo di configurazione di **Item**, ma solo al suo header...

Quindi, come già fatto prima, diamo i comandi per creare l'archivio...

```
$ gcc -c  astrostack.c
$ ar ruv astrostack.a astrostack.o
```

Ora abbiamo due librerie... una per gli **Item** e una per lo stack... creiamoci un file *test.c* per fare alcuni test su di esse..

```
//test.c
#include <stdio.h>
#include <stdlib.h>
#include "item.h"
#include "astrostack.h"

int main(int argc, char **argv)
{
     Item i3 = readItem();
     Item i5 = readItem();
     Item i7 = readItem();
     Item p4;
     AstroStack s = initstack();

     printf("Inizializzazione stack....ok\nPush del 3° valore
inserito...\n");
     push(s,i7);
     printf("Push del 2° valore inserito...\n");
     push(s,i5);
     printf("Push del 1° valore inserito...\n");
     push(s,i3);
     int lung = size(s);
     printf("Lunghezza dello stack dopo 3 push... %d \n",
lung);
     printf("Pop del valore in testa...\n");
     p4 = pop(s);
     printItem(p4);
     printf("Pop del valore in testa...\n");
     p4 = pop(s);
     printItem(p4);
     printf("Pop del valore in testa...\n");
     p4 = pop(s);
     printItem(p4);
     lung =size(s);
     printf("Lunghezza dello stack dopo 3 pop... %d \n",
lung);

     return 0;
}
```

Come potete vedere, anche il nostro *test.c* è completamente svincolato dal tipo di configurazione di **Item**.
Per compilare il nostro programma ci basta dare il seguente comando:

```
$ gcc test.c -o test item-int.a astrostack.a
```

Ed ecco qui il risultato..

```
$ ./test
Insert value to read: 1
Insert value to read: 2
Insert value to read: 3
Inizializzazione stack....ok
Push del 3° valore inserito...
Push del 2° valore inserito...
Push del 1° valore inserito...
Lunghezza dello stack dopo 3 push... 3
Pop del valore in testa...
Value of item: 1
Pop del valore in testa...
Value of item: 2
Pop del valore in testa...
Value of item: 3
Lunghezza dello stack dopo 3 pop... 0
$
```

Provate a cambiare *item-int.a* con *item-string.a*, e vedrete che il programma continuerà a funzionare perfettamente.

Riassumendo...
per creare una libreria avete bisogno di:

- il file *.h* con la dichiarazione dei tipi e dei prototipi delle funzioni
- il file *.c* con il corpo delle funzioni e dei tipi utilizzati dalle funzioni (con l'**include** del file *.h*)

Dunque, create un archivio *.a* contenente Il file *.o* del body della libreria. Fatto questo potete distribuire la vostra libreria, ovvero distribuendo il file *.h* con il suo file *.a*.
È possibile mettere più file *.o* dentro l'archivio, ovviamente ricordatevi di distribuire tutti i vari file *.h* necessari.
Nel programma basterà includere la libreria *.h*.
Se la libreria non si trova nella stessa path del vostro file, vi ricordo che potete specificarla voi. Per esempio:

```
#include "headers/header.h"
```

Potete creare un unico eseguibile anche usando direttamente i file *.o*, per esempio con i file *.o* generati precedentemente possiamo fare:

```
$ gcc -o eseguibile test.o item-int.o astrostack.o
```

Questo è tutto! Buona programmazione C!

Autore: Borsato Claudio