Carlo Gaibisso
Bruno Martino

Programmazione e Laboratorio di Programmazione

Lezione IX

Gestione dinamica della memoria

Carlo Gaibisso
Bruno Martino

Direttiva per il preprocessore

Attenzione!!!

Le librerie del C mettono a disposizione del programmatore un insieme di funzioni per la gestione della memoria. Per utilizzare tali funzioni all'interno di un file è necessario includere in testa allo stesso la direttiva per il preprocessore:

 # include <stdlib.h>

Carlo Gaibisso
Bruno Martino

La funzione malloc()

- **Buffer**

  una sequenza contigua di byte (locazioni) in memoria centrale

- **Il tipo void *:**

  indirizzo di una variabile di tipo non specificato

- **Signature (firma):**

  `void *malloc(size_t size);`

- **Modifiche allo stato della memoria:**

  alloca un buffer di memoria di size byte

- **Valore restituito:**
  - l'indirizzo del primo byte del buffer, in caso di successo
  - il valore NULL altrimenti

Carlo Gaibisso
Bruno Martino

Il tipo size_t

- **Definizione:**

  typedef unsigned int size_t;

  utilizzato per rappresentare dimensioni

- **Range di rappresentazione:**

  tra 0 e $2^{sizeof(size\_t)*8}-1$

- **Attenzione:**

  per problemi di compatibilità tra i diversi compilatori, eviteremo l'uso del tipo size_t

Carlo Gaibisso
Bruno Martino

Operatore di casting

- **Sintassi:**

  (tipo_di_dato) espressione

  con tipo_di_dato identificatore di un tipo predefinito o non

- **Valore:**

  il valore di espressione convertito in un valore di tipo tipo_di_dato

Carlo Gaibisso
Bruno Martino

La funzione free()

- **Signature:**

  void free(void *ptr);

- **Modifiche allo stato della memoria:**

  rilascia il buffer di memoria di indirizzo iniziale ptr allocato da una precedente chiamata della malloc() o calloc()

Carlo Gaibisso
Bruno Martino

Uso delle funzioni malloc() e free()

```
/* sorgente: malloc.c */
/*
** illustra il corretto utilizzo dalle funzioni malloc() e free()
*/

/* inclusione del file di intestazione della libreria standard che
** contiene definizioni di macro, costanti e dichiarazioni di funzioni
** e tipi funzionali alle varie operazioni di I/O */
#include <stdio.h>

/* inclusione del file di intestazione della libreria standard che
** contiene definizioni di macro, costanti e dichiarazioni di funzioni
** di interesse generale */
#include <stdlib.h>

/* funzione che visualizza il contenuto di un buffer di interi */
void VisBuffInt(int *ptr, unsigned int nro_val)
 {
 /* definisce il cursore per la scansione del buffer */
 int curs;

 /* scandisce il buffer visualizzandone il contenuto */
 for (curs = 0; curs <= nro_val-1; curs++)
   printf("\nElemento %d: %d", curs, ptr[curs]);
 }
```

Continua …

Carlo Gaibisso
Bruno Martino

Uso delle funzioni malloc() e free()

Continua …

```
/*
** alloca, e dopo averne visualizzato il contenuto,
** rilascia, un buffer di interi di dimensione specificata
** a run-time
*/
int main()
  {
  /* definisce la variabile per la dimensione del buffer
  ** e per il suo indirizzo iniziale */
  unsigned int nro_val;
  int *ptr;

  /* acquisisce la dimensione del buffer */
  printf("\nQuanti valori? ");
  scanf("%u", &nro_val);

  /* alloca il buffer */
  ptr = (int*) malloc(nro_val * sizeof(int));
```

Continua …

Carlo Gaibisso
Bruno Martino

Uso delle funzioni malloc() e free()

Continua …

```
/* se l'allocazione e' fallita termina */
if (ptr == NULL)
  {
  printf("\nAllocazione fallita");
  return(0);
  };
/* se l'allocazione ha successo visualizza il contenuto
** del buffer e rilascia la memoria per questo allocata */
printf("\nAllocazione avvenuta con successo\n");
printf("\nContenuto del buffer: ");
VisBuffInt(ptr, nro_val);
free(ptr);
return(1);
}
```

Carlo Gaibisso
Bruno Martino

I memory leak

- **Memory leak:**

  area di memoria allocata e non più accessibile non esistendo puntatori che la riferiscono

```
void *ptr = malloc(4);
…
ptr = malloc(1);
…
free(ptr);
```

104

104

ptr

099		x
100		x
101		x
102		x
103		
104		
105		

E ora come recupero le locazioni dalla 099 alla 102?

Carlo Gaibisso
Bruno Martino

La funzione calloc()

- **Signature:**

  void *calloc(size_t nro_var, size_t dim_var);

- **Modifiche allo stato della memoria:**

  alloca un buffer di memoria per nro_var variabili ognuna di dimensione dim_var e le inizializza a 0

- **Valore restituito:**

  - **l'indirizzo del primo byte del buffer, in caso di successo**
  - **il valore NULL altrimenti**

Carlo Gaibisso
Bruno Martino

Uso delle funzioni calloc() e free()

```
/* sorgente: calloc.c */
/* illustra il corretto utilizzo dalla funzione calloc() e della free() */
/* inclusione del file di intestazione della libreria standard che
** contiene definizioni di macro, costanti e dichiarazioni di funzioni
** e tipi funzionali alle varie operazioni di I/O */
#include <stdio.h>
/* inclusione del file di intestazione della libreria standard che
** contiene definizioni di macro, costanti e dichiarazioni di funzioni
** di interesse generale */
#include <stdlib.h>
/* funzione che visualizza il contenuto di un buffer di interi */
void VisBuffInt(int *ptr, unsigned int dim)
  {
  /* definizione della variabile per la scansione del buffer */
  int curs;
  /* scandisce il buffer visualizzandone il contenuto */
  for (curs = 0; curs <= dim-1; curs++)
    printf("\nElemento %d: %d", curs, ptr[curs]);
  }
```

Continua …

Carlo Gaibisso
Bruno Martino

Uso delle funzioni calloc() e free()

Continua …

```
/* alloca, visualizza e successivamente rilascia,  un buffer per
** un numero di valori interi specificato a run-time */
int main()
  {
  /* definisce le variabili per la dimensione del buffer, espressa
  ** in numero di interi, e per il suo indirizzo iniziale */
  unsigned int nro_val;
  int *ptr;

  /* acquisisce la dimensione del buffer */
  printf("\nQuanti valori? ");
  scanf("%u", &nro_val);

  /*  alloca il buffer e inizializza il valore di ogni sua variabile a 0 */
  ptr = (int*) calloc(nro_val, sizeof(int));
```

Continua …

Carlo Gaibisso
Bruno Martino

Uso delle funzioni calloc() e free()

Continua …

```
/* se l'allocazione fallisce termina */
  if (ptr == NULL)
    {
    printf("\nAllocazione fallita");
    return(0)
    };
  /* se l'allocazione ha successo visualizza il contenuto del
  ** buffer e poi libera la memoria per questo allocata */
  printf("\nAllocazione avvenuta con successo\n");
  printf("\nContenuto del buffer:");
  VisBuffIn(ptr, nro_val);
  free(ptr);
  return(1);
  }
```

Carlo Gaibisso
Bruno Martino

La funzione memcpy()

- **Signature:**

  `void * memcpy(void * dest, void * src, size_t n)`

- **Modifiche allo stato della memoria:**

  copia n byte dall'indirizzo src all'indirizzo dest

- **Valore restituito:**

  l'indirizzo dest

- **Direttive per il preprocessore:**

  includere la direttiva `#include <string.h>` per utilizzare la funzione

- **Attenzione:**

  i buffer sorgente (da src per n byte) e destinazione (da dest per n byte) non devono sovrapporsi

Carlo Gaibisso
Bruno Martino

La funzione memcpy()

```
/* sorgente: memcpy.c */
/* illustra il corretto utilizzo dalla funzione memcpy() */
/* inclusione del file di intestazione della libreria standard
** che contiene definizioni di macro, costanti e dichiarazioni
** di funzioni e tipi funzionali alle varie operazioni di I/O */
#include <stdio.h>
/* inclusione dei file di intestazione delle libreria standard che
** contengono definizioni di macro, costanti e dichiarazioni di funzioni
** di interesse generale, e per la gestione della memoria e
** delle stringhe */
#include <stdlib.h>
#include <string.h>
/* funzione che inizializza il contenuto di un buffer con una
** sequenza progressiva di interi a partire da 0 */
void InBuffInt(int *ptr, unsigned int dim)
  {
  /* definisce il cursore per la scansione del buffer */
  int curs;
  /* scandisce il buffer inizializzandone il contenuto */
  for (curs = 0; curs <= dim-1; curs++)
    ptr[curs] = curs;
  }
```

Continua …

Carlo Gaibisso
Bruno Martino

La funzione memcpy()

```
/* funzione che visualizza il contenuto di un buffer di interi */
void VisBuffInt(int *ptr, unsigned int dim)
  {
  /* definisce il cursore per la scansione del buffer */
  int curs;
  /* scandisce il buffer visualizzandone il contenuto */
  for (curs = 0; curs <= dim-1; curs++)
    printf("\nElemento %d: %d", curs, ptr[curs]);
  };
/* alloca due buffer di interi, inizializza il I e lo visualizza,
** lo copia nel II e poi ne visualizza il contenuto. Infine rilascia
** la memoria allocata per entrambi i buffer */
int main()
  {
  /* definisce le variabili per la dimensione dei buffer,
  ** espressa in numero di interi, e per i loro indirizzi
  ** di inizio */
  unsigned int nro_val;
  int *ptr_1, *ptr_2;

  /* acquisisce la dimensione dei buffer */
  printf("\nQuanti valori nei buffer? ");
  scanf("%u", &nro_val);
```

Continua …

Continua …

Carlo Gaibisso
Bruno Martino

La funzione memcpy()

Continua …

```
/* alloca i due buffer */
ptr_1 = (int *) calloc(nro_val, sizeof(int));
ptr_2 = (int *) calloc(nro_val, sizeof(int));
/* controlla se l'allocazione ha successo */
if ((ptr_1 == NULL) || (ptr_2 == NULL))
  {
  /* se fallisce rilascia la memoria eventualmente allocata
  ** e termina */
  printf("\nAllocazione fallita");
  if (ptr_1 != NULL)
    free(ptr_1);
  if (ptr_2 != NULL)
    free(ptr_2);
  return(0);
  };
/* altrimenti inizializza il I buffer e lo visualizza */
printf("\nAllocazione avvenuta con successo\n");
InBuffInt(ptr_1, nro_val);
printf("\nContenuto I Buffer:");
VisBuffInt(ptr_1, nro_val);
```

Continua …

Carlo Gaibisso
Bruno Martino

La funzione memcpy()

```
/* copia il I buffer nel II e lo visualizza */
ptr_2 = memcpy(ptr_2, ptr_1, nro_val*sizeof(int));
printf("\nContenuto II Buffer:");
VisBuffInt(ptr_2, nro_val);
/* rilascia la memoria allocata per entrambi i buffer */
free(ptr_1);
free(ptr_2);
return(1);
}
```