Programmazione I (Tucci/Distasi)

PR1 MT/RD 16/07/2021-PRES

Modello: 1

Cognome: _____

Nome: _____

Matricola: _____

Email: _____

Regole del gioco: Compilare i dati personali prima d'incominciare. Una volta iniziata la prova, non è consentito lasciare l'aula. Usare questi stessi fogli (compreso il retro, dove necessario) per rispondere. *Buon lavoro!*

1. Consideriamo la seguente struttura dati, che rappresenta un rettangolo.

```
typedef struct
{
  double x;                    // punto in alto a sinistra, coordinata x
  double y;                    // punto in alto a sinistra, coordinata y
  double w;                    // larghezza del rettangolo
  double h;                    // altezza del rettangolo
} Rect;
```

   - Scrivere una funzione C

     `int rect_cmp(Rect *r1, Rect *r2)`

     che riceve come parametri gli indirizzi di due rettangoli e segnala quale dei due ha l'area più grande, restituendo: -1 se è il primo che ha l'area più grande; 1 se invece è il secondo; 0 se le due aree sono equivalenti.

   - Scrivere una funzione C

     `Rect * rect_smaller(Rect *A, int n, Rect *toobig, int *newsize)`

     che riceve come parametri un array `A[]` di `n` rettangoli e un puntatore `toobig` a un rettangolo. La funzione restituisce un array allocato dinamicamente contenente copie di tutti i rettangoli in `A[]` che hanno area minore del rettangolo puntato da `toobig`. Il parametro output `newsize` sarà impostato alla dimensione dell'array risultante. Nel caso nessun rettangolo in `A[]` soddisfi la condizione, `rect_smaller()` restituirà `NULL` e imposterà `newsize` a zero.

2. Scrivere una funzione C

```
int rect_save_smaller(FILE *savefile, Rect *A, int n, Rect *r0)
```

che riceve come parametri un file binario già aperto, un array di rettangoli `A[]` con la sua dimensione `n`, e un puntatore a rettangolo `r0`. La funzione salva nel file tutti i rettangoli in `A[]` che hanno un'area minore di quella di `r0` e restituisce il numero di rettangoli salvati.

Risposte per il modello 1

1. Consideriamo la seguente struttura dati, che rappresenta un rettangolo.

```
typedef struct
{
  double x;                    // punto in alto a sinistra, coordinata x
  double y;                    // punto in alto a sinistra, coordinata y
  double w;                    // larghezza del rettangolo
  double h;                    // altezza del rettangolo
} Rect;
```

   - Scrivere una funzione C

     ```
     int rect_cmp(Rect *r1, Rect *r2)
     ```

     che riceve come parametri gli indirizzi di due rettangoli e segnala quale dei due ha l'area più grande, restituendo: -1 se è il primo che ha l'area più grande; 1 se invece è il secondo; 0 se le due aree sono equivalenti.

     Risposta
     Ecco una possibile soluzione.

```
double area(Rect * r)
{
  return r->w * r->h;
}

int rect_cmp(Rect * r1, Rect * r2)
{
  double diff;

  diff = area(r2) - area(r1);
  if (diff > 0)                // r2 larger
    return 1;
  else if (diff < 0)           // r1 larger
    return -1;
  else
    return 0;                  // equal areas
}

Rect *rect_cpy(Rect * dest, Rect * src)
{
  dest->x = src->x;
  dest->y = src->y;
  dest->w = src->w;
  dest->h = src->h;
  return dest;
}
```

- Scrivere una funzione C

  ```
  Rect * rect_smaller(Rect *A, int n, Rect *toobig, int *newsize)
  ```

  che riceve come parametri un array `A[]` di `n` rettangoli e un puntatore `toobig` a un rettangolo. La funzione restituisce un array allocato dinamicamente contenente copie di tutti i rettangoli in `A[]` che hanno area minore del rettangolo puntato da `toobig`. Il parametro output `newsize` sarà impostato alla dimensione dell'array risultante. Nel caso nessun rettangolo in `A[]` soddisfi la condizione, `rect_smaller()` restituirà `NULL` e imposterà `newsize` a zero.

Risposta
Ecco una possibile soluzione.

```
void *xmalloc(size_t nbytes)    // malloc() con controllo errore
{
  void *result;

  if ((result = malloc(nbytes)) == NULL)
    {
      fprintf(stderr, "malloc(%lu) failed. Exiting.\n", nbytes);
      exit(-1);
    }
  return result;
}

Rect *rect_smaller(Rect * A, int n, Rect * toobig, int *newsize)
{
  int i, j, count = 0;
  Rect *result;

  for (i = 0; i < n; i++)       // count "small" rects
    {
      if (rect_cmp(&A[i], toobig) == 1)
        {
          count++;
        }
    }
  *newsize = count;             // set output parameter
  if (count == 0)               // no "small" rect to be copied
    {
      return NULL;
    }
  // else, allocate result array and copy small rects
  result = xmalloc(sizeof(Rect) * count);
  for (i = 0, j = 0; i < n; i++)
    {
      if (rect_cmp(&A[i], toobig) == 1)
        {
          rect_cpy(&result[j], &A[i]);
          j++;
        }
    }
  return result;
}
```

2. Scrivere una funzione C

```
int rect_save_smaller(FILE *savefile, Rect *A, int n, Rect *r0)
```

che riceve come parametri un file binario già aperto, un array di rettangoli `A[]` con la sua dimensione `n`, e un puntatore a rettangolo `r0`. La funzione salva nel file tutti i rettangoli in `A[]` che hanno un'area minore di quella di `r0` e restituisce il numero di rettangoli salvati.

Risposta
Ecco una possibile soluzione.

```
int rect_save_smaller(FILE * savefile, Rect * A, int n, Rect * r0)
{
  Rect *saveus;
  int savecount;
  int status;

  saveus = rect_smaller(A, n, r0, &savecount);  // build array to be saved
  if (savecount != 0)           // only if array not empty
    {
      status = fwrite(&saveus[0], sizeof(Rect), savecount, savefile);
      if (status != savecount)
        {
          fprintf(stderr, "Couldn't save rectangles. Leaving.\n");
          exit(-2);
        }
      free(saveus);             // not needed anymore
    }
  return savecount;
}
```