Laboratorio
(Attività Preparatoria)

Sommario: 9 Marzo, 2018

- Esercizi Propedeutici:
  - Espressività del Linguaggio.
    Struttura di un Programma: Versionamento
  - Allocazione Dinamica
    Quando, Come, Perchè: Studio di un caso

Espressività del Linguaggio

- **Struttura di un Programma: Versionamento**.
  - Siamo Partiti da un L. di Programmazione e da un algoritmo, che ci sono stati "imposti", per:
    - Un problema a cui dare una soluzione automatizzata
    - Una funzione calcolabile di cui fornire un programma
    - Una Computer Application da realizzare

    3 modi di esprimere l'Ordinamento in C con QuickSort
  - Alla fine ciascuno di noi ha fornito un Programma in C che risponde alla richiesta.
  - Ottenendo ben 22 versioni diverse che differiscono tra loro:
    - nei costrutti utilizzati per definire le principali fasi del procedimento;
    - nelle strutture dei dati utilizzati;
    - nello spazio dei nomi introdotti (variabili, costanti)
  - Cosa considerare per confrontare le diverse versioni e
    - individuare la migliore ?
    - e/o migliorare ancora la versione scelta?

Attività

- **Discutere ed Individuare dei criteri di confronto;.**
  soluzione (discussa in aula)
  Leggibilità intesa come capacità di:
  - mostrare le fasi essenziali del procedimento (stiamo usando programmazione prescrittiva mediante un linguaggio imperativo);
  - localizzare il codice di ogni fase in procedure;
  - usare una struttura di codice gerarchica che mostri dettagli di ogni fase solo in profondità (come approfondimento della relativa definizione)

- **Applicare i criteri di confronto scelti alle versioni QSort2 e QSort3** (vedi ListingA2.1 e/o figura slide sotto)
  soluzione discussa in aula ...

- **Discutere possibili miglioramenti della versione individuata come migliore.**
  soluzione discussa in aula ...

Attività

QS2

```c
int N = 15; // Parametro di programma (non modificabile)
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici crescenti in [Left..Right]
    int J=Right; //indici decrescenti in [Left..Right]
    int temp=A[(Left+Right)/2];  //elemento di separazione
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            int scambia=A[I];
            A[I]=A[J];
            A[J]=scambia;
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    // Allocaione Sequenza da ordinare
    int Seq[N];
    int k;
    //lettura sequenza k<=N interi da ordinare;
    int i=0;
    while (i<N && scanf("%d",&Seq[i])==1) i++;
    k=i-1; //-- K+1: size effettiva della sequenza
     //ordinamento QuickSort
    QuickSort(0,k,Seq);
    //stampa sequenza ordinata;
    for (int i=0; i<k; i++) printf("%d,",Seq[i]);
    printf("%d\n",Seq[k]);
    return(1);
}
```

QS3

```c
int N = 15; // Parametro di programma (non modificabile)
void Scambia(int *x, int *y){
    int scambia = *x;
    *x = *y;
    *y = scambia;
}
void leggiSeq(int Seq[], int *upIndex){
    int i = 0;
    while (i<N && scanf("%d",&Seq[i])==1)i++;
    *upIndex = i-1;
}
void stampaSeq(int Seq[], int upIndex){
    for (int i=0; i<upIndex; i++) printf("%d,",Seq[i]);
    printf("%d\n",Seq[upIndex]);
}
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    int temp=A[(Left+Right)/2];
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            Scambia(&A[I],&A[J]);
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    int Seq[N];
    int k;
    leggiSeq(Seq,&k);     //lettura sequenza
    QuickSort(0,k,Seq);   //ordinamento
    stampaSeq(Seq,k);     //stampa sequenza
    return(1);
}
```

Allocazione Dinamica e Moduli in C

Esercizio (semplificato)

Scrivere un programma C che introduca una implementazione per i parseTree che abbiamo visto nella lezione del 8/3/2018. Allo scopo si fornisca l'implementazione di 2 nuovi tipi di dato e delle operazioni a loro applicabili ... Si completi il programma con opportuni casi di test.

Esercizio (Completo)

Scrivere un programma C che calcoli la funzione $\Rightarrow^{\texttt{Tree}}$ *(deriva) su parse tree, data una grammatica libera (vedi lucidi relativi). Allo scopo, si completi il testo con le assunzioni e i vincoli che si ritenga necessari apportare.*
Si fornisca infine, il codice per applicare il programma ad un caso concreto, quale la costruzione del parse tree ottenuto dalla derivazione dell'espressione $3 \star \texttt{x} + 10$ *con la grammatica vista a lezione, utilizzando l'opportuno lessico.*

Attività

- **Riesaminare proprietà dei parseTree** Definizione, Uso, anche rispetto alle operazioni richieste (vedi ListingA2.2 e/o figura slide sotto)
  soluzione (discussa in aula)...

- **Definire una struttura per la rappresentazione dei parseTree**. Definizione prima astratta, poi formale in C
  soluzione (discussa in aula)...

- **Definire ciascuna operazione richiesta.** Fornendo ed applicando le soluzioni parziali a casi di test durante l'intero sviluppo.
  soluzione da completare e discutere ...

Attività

parseTree.h e' un modulo che fornisce l'implementazione di 3 nuovi tipi di dato e la specifica delle operazioni applicabili.

- Il tipo **parseTree** con le seguenti operazioni (di interfaccia):
  - makeEmpty
  - makeLeaf
  - makeRooted
  - isEmpty
  - isLeaf
  - isRooted
  - getLabel
  - getSons
  - printlnTree
- Il tipo **treeList** con le seguenti operazioni (di interfaccia)
  - nil
  - cons
  - split
- Il tipo **bool** senza operazioni con due valori: {`false`, `true`}