Caso di studio: lo stack

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione

a.a. 2012/2013

Section 1

Code e Pile

Tipi di dato astratti

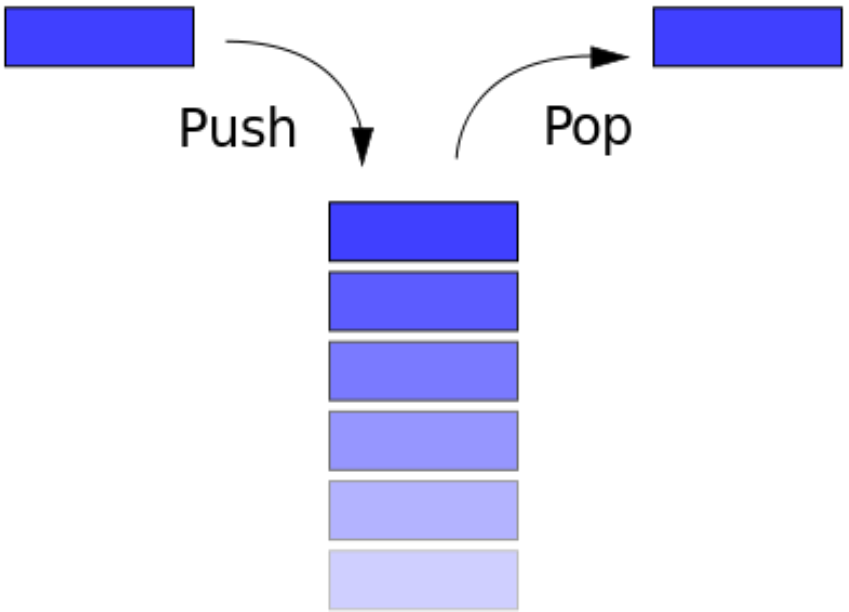
Abstract data type

Un tipo di dato astratto (ADT) è un tipo di dato per il quale si stabiliscono delle modalità di manipolazione (accesso e/o cambiamento) indipendentemente dalla sua implementazione

- L'idea è di separare l'uso di un dato dalla sua rappresentazione
- Astrazione sofisticata (corrispondenti in algebra?)

Primo esempio di ADT: lo stack (pila)

- Lo stack è una collezione di item dello stesso tipo sulla quale sono definite le seguenti operazioni:
  - push: inserisce un item nello stack
  - pop: estrae un item dallo stack, rimuovendolo. L'item da estrarre è quello inserito più recentemente secondo la regole Last-In-First-Out (LIFO)
  - is_empty: decide se uno stack è vuoto



Secondo esempio di ADT: la queue (coda)

- La coda è una collezione di item dello stesso tipo sulla quale sono definite le seguenti operazioni:
  - enqueue: inserisce un item nella coda
  - dequeue: estrae un item dalla coda, rimuovendolo. L'item da estrarre è quello inserito sa più tempo secondo la regole First-In-First-Out (FIFO)
  - is_empty: decide se una coda è vuota

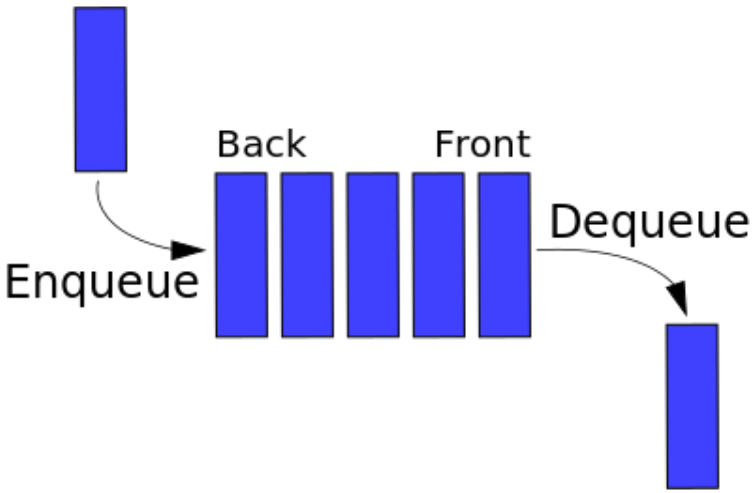


Image taken from http://it.wikipedia.org/wiki/File:Data_Queue.svg

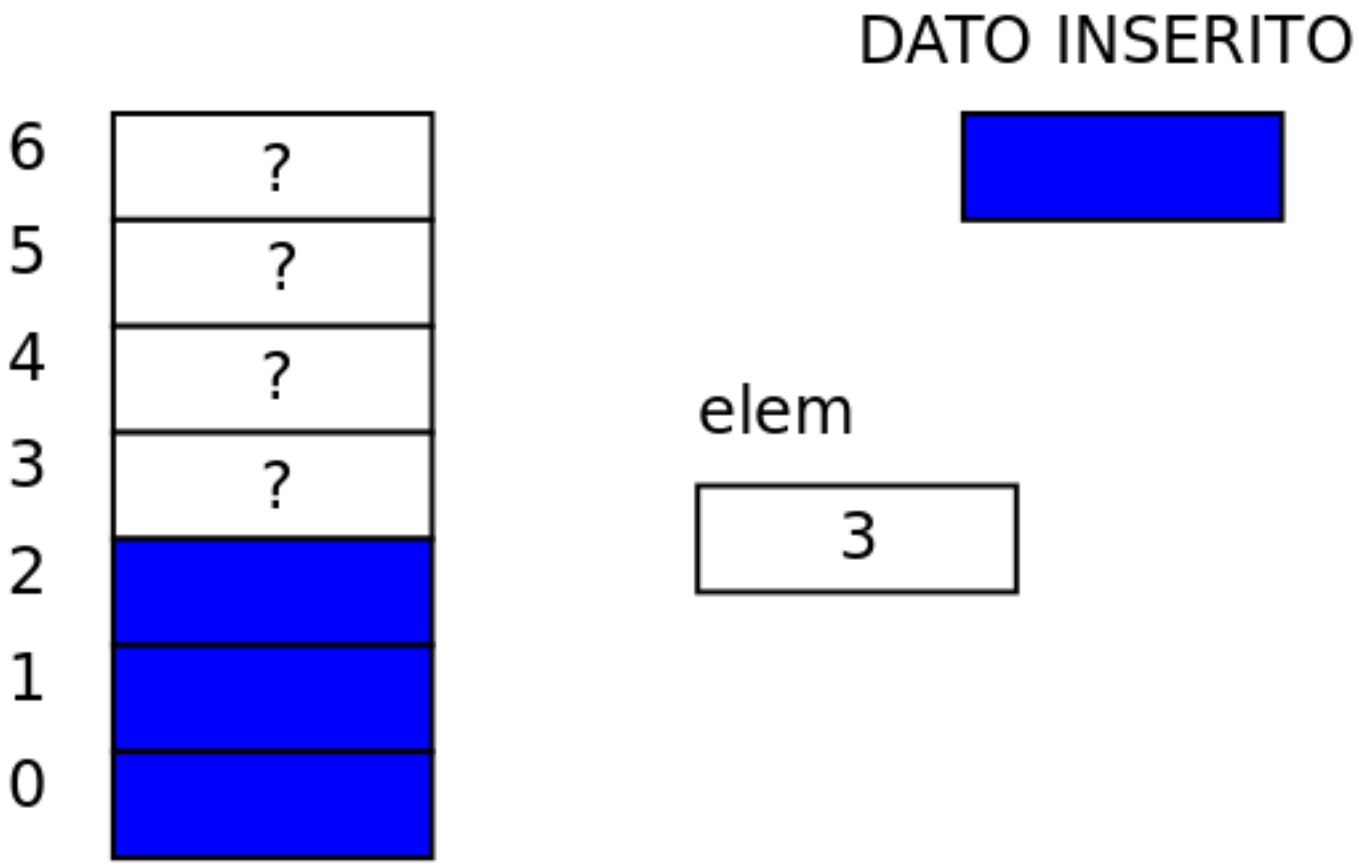
Section 2

Implementazione dello stack: prima vesione

Implementare un ADT: primi passi

- Innanzitutto decidiamo come memorizzare la struttura che ci interessa
- L'idea più semplice è quella di usare un vettore con una dimensione massima *abbastanza* grande di celle il cui tipo corrisponde al tipo degli item da memorizzare
- Inoltre usiamo una variabile che memorizza il numero di elementi inseriti nello stack
- Questa variabile, incidentalmente, memorizza anche la prossima posizione libera dello stack

Implementazione dello stack



Dichiarazione dei tipi (stack di float)

```
#define MAXDIM 100

float stack_float[MAXDIM];
int elem=0; /*stack vuoto*/
```

Prototipi delle funzioni

```
/*return true se operazione ok*/
int push(float vet[], int* ptop, float item);

/*return true se operazione ok*/
int pop(float vet[], int* ptop, float* pitem);

/*return true se empty stack*/
int is_empty(int top);
```

Implementazione della push

```
int push(float vet[], int* ptop, float item) {
    if (*ptop < MAXDIM) { /*ok spazio*/
        vet[*ptop] = item;
        (*ptop)++;
        return 1;
    }
    else /* stack overflow */
        return 0;
}
```

Implementazione della pop

```
int pop(float vet[], int* ptop, float* pitem) {
    if (*ptop > 0) { /*stack non vuoto*/
        (*ptop)--;
        *pitem = vet[*ptop];
        return 1;
    }
    else /*stack underflow*/
        return 0;
}
```

Implementazione della is_empty

```
int is_empty(int top) {
    return (top==0);
}
```

Esempio d'uso

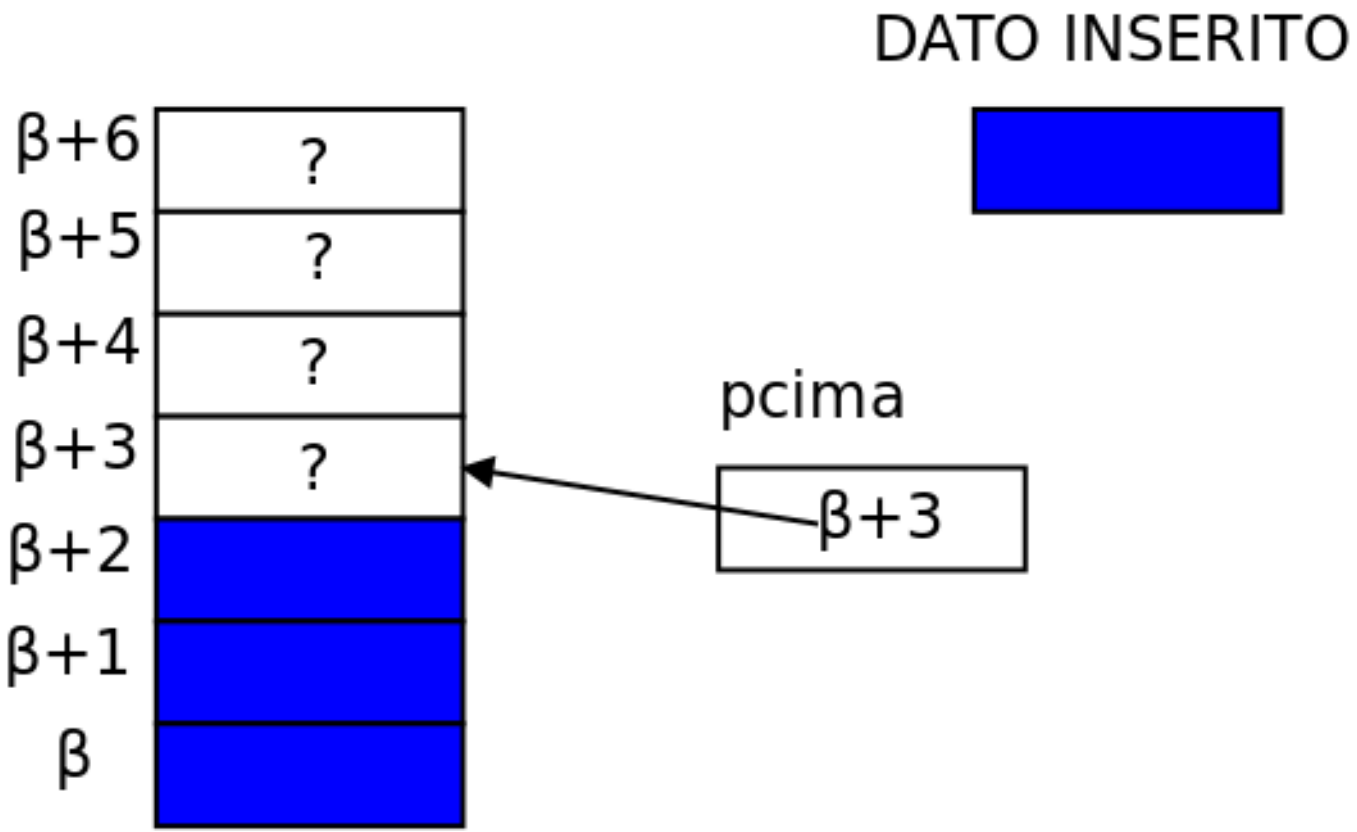
- Leggere una sequenza di interi che termini con lo 0 e stamparla al contrario

```
int main() {
    float miostack[MAXDIM];
    int cima = 0;
    float lettura;
    do {
        scanf(``%f'', &lettura);
    } while (push(miostack, &cima, lettura) && lettura!=0.0);
    /*ripete l'acquisizione fintanto che c'e' spazio nello stack*/
    /*oppure incontra il numero 0*/
    while (!is_empty(cima)) {
        /*non mi interessa il valore ritornato dalla pop*/
        pop(miostack, &cima, &lettura);
        printf(`` %f '', lettura);
    }
    return 0;
}
```

Un altro punto di vista

- Proponiamo un'alta implementazione sempre basata su vettori
  - `float miostack[MAXDIM];`
- Rispetto alla precedente implementazione la prima cella libera è puntata da una variabile puntatore
  - `float* pcima;`
- Il numero di elementi nello stack è dato dall'operazione tra indirizzi:
  - `pcima - miostack` equivalente a
  - `pcima - &miostack[0]`

Implementazione dello stack



Modificare il puntatore alla cima

- Le operazione push e pop devono modificare il puntatore alla cima
- Per C è possibile definire il puntatore a puntatore al tipo `type`, cioè: `type**`
- Gli operatori & e * funzionano in modo analogo a quanto visto
  - `int **ppc; int *pc; int pippo;`
  - `ppc = &pc;`
  - `pc = &pippo;` oppure `*ppc = &pippo;`
  - `**ppc=7;` o `*pc=7;` o `pippo=7;`

Prototipi delle funzioni

```
int push(float vet[], float **ppcima, float item);

int pop(float vet[], float **ppcima, float *pitem);

int is_empty(float vet[], float *pcima);
```

Implementazione funzione push

```
int push(float vet[], float **ppcima, float item) {
    if (*ppcima < vet + MAXDIM) {
        **ppcima = item;
        (*ppcima)++;
        return 1;
    }
    else
        return 0;
}
```

Implementazione della funzione pop

```
int pop(float vet[], float **ppcima, float *pitem) {
    if (*ppcima > vet) {
        (*ppcima)--;
        *pitem = **ppcima;
        return 1;
    }
    else
        return 0;
}
```

Implementazione is_empty

```
int is_empty(float vet[], float* pcima) {
    return (pcima == vet);
}
```

Esempio d'uso

- Leggere una sequenza di interi che termini con lo 0 e stamparla al contrario

```
int main() {
    float miostack[MAXDIM];
    float* cima = miostack;
    float lettura;
    do {
        scanf(``%f'', &lettura);
    } while (push(miostack, &cima, lettura) && lettura!=0.0);
    /*ripete l'acquisizione fintanto che c'e' spazio nello stack*/
    /*oppure incontra il numero 0*/
    while (!is_empty(miostack, cima)) {
        /*non mi interessa il valore ritornato dalla pop*/
        pop(miostack, &cima, &lettura);
        printf(`` %f '', lettura);
    }
    return 0;
}
```

Osservazioni finali

- Abbiamo dato due implementazioni diverse di un ADT
- Tuttavia il main non è completamente indipendente dall'implementazione scelta
- Linguaggi di programmazione più evoluti consentono la completa indipendenza tra il codice che usa l'ADT e quello che lo implementa
  - C++, Java, C#, ...

Esercizi

- Implementare l'ADT coda di int usanto array e indici e usando array e puntatori
  - scrivere un main di prova