La Struttura Dati Pila

La pila o stack è una particolare struttura dati in cui l'inserimento e la cancellazione sono consentite solo in una specifica posizione, chiamata *cima* (*top*) dello stack. La gestione dello stack è anche chiamata LIFO (Last In First Out), in quanto l'ultimo elemento inserito è il primo ad essere estratto; come detto l'ultimo elemento inserito (che potrà essere anche estratto) occupa una posizione denominata **top** o **cima**.

Sia data una pila composta dagli elementi a_1, a_2, a_3, …., a_n tutti omogenei (osia appartenenti allo stesso tipo di dato) e si supponga che la cima sia rappresentata dal primo elemento a_1. L'inserimento di un elemento x produce la nuova pila x, a_1, a_2, a_3, …., a_n. La nuova cima della PILA è relativa all'elemento appena inserito, ossia x. La cancellazione della cima ripristina la pila iniziale a_1, a_2, a_3, …., a_n.

1.Operazioni Primitive sulla Struttura Dati Pila

La struttura dati Pila è caratterizzata da alcune operazioni fondamentali. Sia:

- ***s*** una generica pila
- ***x*** un generico elemento dello stesso tipo degli elementi presenti nella pila. Nel seguito il tipo degli elementi contenuti in uno stack verrà chiamato **tipobaseStack.**

Le seguenti operazioni primitive vengono definite per un ADT Pila:

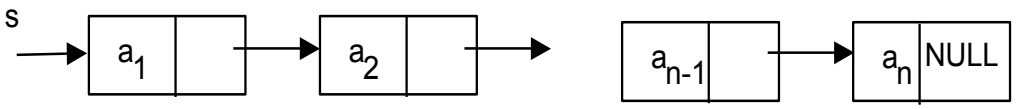
- ***MakeNullStack(s)***. La funzione inizializza una Pila, restituendo una Pila vuota.
- ***EmptyStack(s)***. La funzione restituisce il valore logico vero se la Pila ***s*** è vuota.
- ***FullStack(s).*** La funzione restituisce il valore logico vero se la Pila ***s*** è piena.
- ***Push(s,x).*** Questa funzione inserisce l'elemento ***x*** nella pila ***s***, nella posizione rappresentata dal top. Se ***s*** è la Pila a_1, a_2,…, a_n, e se il top è rappresentato dall'elemento a_1 della lista, la funzione ***Push(s,x)*** restituisce la nuova pila x, a_1, a_2,…, a_n.
- ***Pop(s).*** Questa funzione elimina l'elemento della pila che rappresenta la cima. Se ***s*** è la Pila a_1, a_2,…, a_n e la cima è rappresentata dalla posizione del primo elemento, la funzione ***Pop(s)*** restituisce la nuova pila a_2, a_3,…, a_n e la nuova cima è rappresentata dalla posizione dell'elemento a_2.
- ***Top(s)***. La funzione restituisce l'elemento in cima alla Pila, che rimane inalterata.

2.Rappresentazione della Struttura Dati Pila.

La realizzazione della struttura dati Pila in linguaggio C può avvenire in diversi modi, ad esempio utilizzando vettori oppure puntatori. Nel seguito verrà analizzata unicamente la realizzazione tramite puntatori.

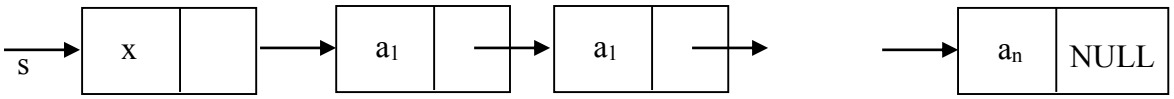
La rappresentazione con puntatori si basa sull'idea di considerare elementi allocati dinamicamente in memoria RAM (Heap), ciascuno dei quali è composto da due campi. Un campo memorizza il generico elemento della struttura dati Pila, mentre l'altro campo memorizza l'indirizzo di memoria del successivo elemento.

La rappresentazione collegata della pila con puntatori può essere simile a quella mostrata in figura.

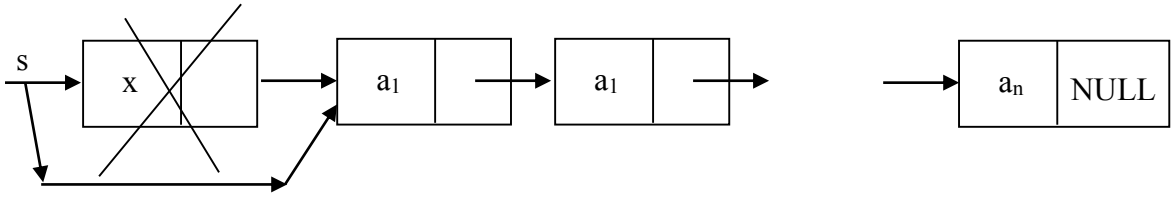


Il puntatore s contiene l'indirizzo dell'ultimo elemento inserito, ossia della cima. Questo elemento a sua volta punta all'elemento inserito al passo precedente e così via. L'elemento (a_n), inserito per primo, non punta a nessun altro elemento (campo puntatore NULL).

L'inserimento di un nuovo elemento, x, consiste semplicemente nell'allocazione di un nuovo elemento in memoria Heap, che dovrà contenere x e che dovrà puntare all'elemento a_1, che rappresentava in precedenza la cima della Pila. La variabile s adesso punterà al nuovo elemento inserito, come visibile dalla seguente figura.

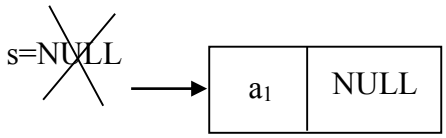


L'eliminazione della cima, analogamente avviene deallocando l'elemento contente la cima della Pila e facendo puntare s all'elemento il cui indirizzo era contentuo nell'elemento di cima, come si evince dalla seguente figura.



Nel caso di una pila vuota, la variabile s dovrà essere uguale a NULL, ad indicare una tale situazione. L'inserimento del primo elemento in Pila, avverrà come detto prima, ossia allocando il nuovo elemento e ponendo s uguale all'indirizzo dell'elemento appena allocato. Tale elemento non

punterà a nessun altro, e dunque avrà il valore NULL nel campo relativo all'indirizzo del prossimo elemento. La figura seguente mostra la condizione in cui la Pila è vuota e si inserisce il primo elemento.



Nel seguito, la rappresentazione Pila con puntatori verrà definita nel linguaggio C. Prima è necessario definire il tipo **tipobaseStack** per rappresentare il tipo di dato di ogni singolo elemento contenuto nella pila. Ad esempio se il tipo di dato di ogni singolo elemento fosse int, l'implementazione di tipobaseStack risulterebbe:

```
typedef int tipobaseStack;
```

Poi per comodità viene definito il tipo **boolean**:

```
typedef short boolean;
```

Deve a questo punto essere definito il generico elemento della struttura dati pila, tramite la definizione dello **struct nodoStack**, che è costituito da campo info che contiene un generico elemento della pila e dal campo next che indica il successivo elemento nella pila.

```
struct nodoStack {
      tipobaseStack info;
      struct nodoStack *next;
};
```

Infine deve essere definito il tipo **Stack**, ossia la definizione di pila che coinciderà con il puntatore all'elemento di tipo struct nodoStack. La sua implementazione è:

```
typedef struct nodoStack * stack;
```

Rappresentazione delle Operazioni Primitive

Le operazioni sulla pila vengono così realizzate:

```
void MakeNullStack(stack *s) {
  *s=NULL;
}

boolean FullStack(stack s) {
  struct nodoStack *temp;

  temp=(struct nodoStack *)malloc(sizeof(struct nodoStack));
  if(temp==NULL) return 1;
  free(temp);
  return 0;
}

boolean EmptyStack(stack s) {
  return(s==NULL);
}

void Push(stack *s, tipobaseStack x) {
  struct nodoStack * temp;

  temp=(struct nodoStack *)malloc(sizeof(struct nodoStack));
  temp->info=x;
  temp->next=*s;
  *s=temp;
}

void Pop(stack *s) {
  struct nodoStack * temp;

  if (!EmptyStack(*s)) {
      temp=(*s)->next;
      free (*s);
      *s=temp;
  }
}

tipobaseStack Top(stack s)
{
  if (!EmptyStack(s))
      return(s->info);
}
```

3. Scrittura di un Programma di Gestione di Stack.

Il seguente programma si riferisce alla gestione delle funzioni principali di una struttura dati pila tramite un menu di comandi. Il programma presuppone che ogni informazione contenuta nella pila sia uno **struct tipobaseStack** composto dai campi: cognome, nome, età.

```
#include<stdio.h>
#include<stdlib.h>

#define FFLUSH while(getchar()!='\n')

#define S 15

typedef struct {
     char cognome[S], nome[S];
     unsigned short eta;
}tipobaseStack;

void LeggiStringa(char s[],unsigned long dim){
  unsigned long i=0;

  for (i=0; i<dim-1;i++)
       if ((s[i]=getchar())=='\n') break;
  if (i==dim-1) FFLUSH;
  s[i]='\0';
}

void LeggiElementoStack(tipobaseStack * p){
     printf("\nInserisci il Cognome ");
     LeggiStringa(p->cognome, S);
     printf("\nInserisci il Nome ");
     LeggiStringa(p->nome, S);
     printf("\nInserisci l'eta' ");
     scanf("%hu",&p->eta);
     FFLUSH;
}

void VisualizzaElementoStack(tipobaseStack x){
     printf("\nCognome = %s ", x.cognome);
     printf("\nNome = %s ", x.nome);
     printf("\nEta' = %d \n\n",x.eta);
}


/*Definizione Pila */

typedef short boolean;

struct nodoStack {
     tipobaseStack info;
     struct nodoStack *next;
};

typedef struct nodoStack *stack;
```

```
void MakeNullStack(stack *s) {
  *s=NULL;
}

boolean FullStack(stack s) {
  struct nodoStack *temp;

  temp=(struct nodoStack *)malloc(sizeof(struct nodoStack));
  if(temp==NULL) return 1;
  free(temp);
  return 0;
}

boolean EmptyStack(stack s) {
  return(s==NULL);
}

void Push(stack *s, tipobaseStack x) {
  struct nodoStack * temp;

  temp=(struct nodoStack *)malloc(sizeof(struct nodoStack));
  temp->info=x;
  temp->next=*s;
  *s=temp;
}

void Pop(stack *s) {
  struct nodoStack * temp;

  if (!EmptyStack(*s)) {
     temp=(*s)->next;
     free (*s);
     *s=temp;
  }
}

tipobaseStack Top(stack s)
{
  if (!EmptyStack(s))
      return(s->info);
}

/*Programma di Gestione Pila */

stack archivio;
short s;
tipobaseStack elem;

int main(void)
{

   MakeNullStack(&archivio);
   do {
      printf("\nMenu di Operazioni \n");
      printf("\n1-Inserimento ");
      printf("\n2-Cancellazione ");
      printf("\n3-Svuota Pila ");
      printf("\n4-Ispeziona Elemento in Testa ");
      printf("\n5-Fine ");
```

```
      printf("\nInserisci la scelta ");
      scanf("%d",&s);
      FFLUSH;

      switch(s){
            case 1 :
                  if (!FullStack(archivio)) {
                        printf("\nInserisci Elemento nello Stack ");
                        LeggiElementoStack(&elem);
                        Push(&archivio,elem);
                  } else printf("\n Pila Piena \n");
                  break;

            case 2 :
                  if (EmptyStack(archivio)) printf("\n Pila Vuota \n");
                  else {
                        Pop(&archivio);
                        printf("\n Primo Elemento Estratto \n");
                  }
                  break;

            case 3 :
                  while (!EmptyStack(archivio)) Pop(&archivio);
                  break;

            case 4 :
                  if (EmptyStack(archivio)) printf("Pila Vuota \n");
                  else {
                    printf("\nElemento in Cima allo Stack ");
                    VisualizzaElementoStack(Top(archivio));
                  }
      }
   } while (s<5);
}
```