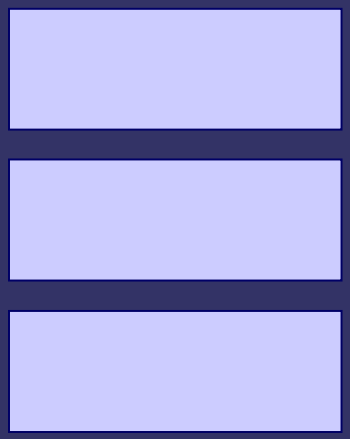
Pile

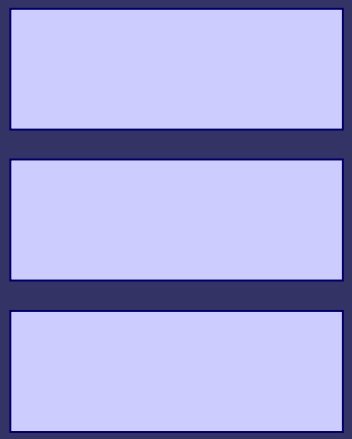
PILA (Stack)

- Una [illegible] un [illegible] ordinato, non limitato, di ele[illegible]dello ste[illegible]o su cui il primo elemento m[illegible]zato e' l[illegible]mento ad essere recuperato ([illegible]-Last Inpu[illegible]utput)



PILA (Stack)

- Una pila è un insieme ordi[illegible]on [illegible] di elementi dello stesso tipo[illegible] il prim[illegible]ento memorizzato e' l'ultimo [illegible]to ad es[illegible]erato (LIFO=Last Input Firs[illegible]t)



PILA (Stack)

- Il tipo PILA è un ADT <S,F,C> dove
  - S = {pila, atomo, boolean}

  pila è il dominio di interesse atomo è il dominio degli elementi che formano le liste

  F = {push, top, empty, pop}

push : atomo x pila -> pila	inserisce un elemento in cima alla pila
top : pila -> atomo	ritorna l'elemento in cima alla pila
empty : pila -> boolean	ritorna il valore vero se la pila è vuota
pop : pila -> pila	ritorna la pila privata dell'elemento in cima

C = pila vuota, è la costante che denota la coda priva di elementi

Implementazione sequenziale di pile

```
typedef int TAtomo;
typedef struct StELem {
                              TAtomo *e;
                              int primo, NMax;
                              } Pila;
int InizializzaPila( Pila *PP, int NElemMax )
{  PP->e=(Tatomo *) malloc(sizeof(TAtomo)*NElemMax);
   if( PP->e == NULL ) return 0;
   PP -> primo = -1;
   return PP->NMax = NElemMax;
  }

int empty( Pila P )
{  return (P.primo == -1);}
```

Implementazione sequenziale di pile

```
int full( Pila P ) {
        return (P.primo == P.NMax-1);
}

int pop ( Pila *PP ) {
        if( !null(*PP) ) return 0;
        PC-> primo--
        return 1;
}

TAtomo  top( Pila P ) {
        if( null(P) ) return NULL;
        return P.e[P.primo];
}
```

Implementazione sequenziale di pile

```
int push (Pila *PP, TAtomo A )
{
        if( full(*PP)) return 0;
        if( null(*PP)) PP -> primo = 0;
        else PP->primo++;
        PP->e[PP->primo] = A;
        return 1;
    }
```

Implementazione concatenata di pile

```
typedef int TAtomo;
typedef struct Stelem {
                    struct Selem *next;
                    TAtomo info;
          } elemPila, *TPila;

int pop (TPila *PP){
    TPila aux;
    if (null(*PP)) return 0;
    aux = *PP;
    *PP = *PP->next;
    free(aux);
     return 1;
}
```

Implementazione concatenata di pile

```
int null( TPila P ) { return P == NULL;}
TAtomo top( Pila P ) {
        if( null(P) ) return NULL;
        return P -> info;
}
int push( TPila *PP, TAtomo A ) {
        Tpila *aux;
        aux = (TPila)malloc(sizeof(elempila));
        if (!aux) return 0;
        aux ->info = A;
        aux -> next = *PP;
        *PP = aux;
        return 1;
}
```