# Esercizio 1 – Stack

**Conversione decimale-binario mediante stack**

- Si realizzi un programma che, ***utilizzando uno stack***, converta un numero decimale nella corrispondente rappresentazione binaria
- Il risultato deve essere ***salvato in una stringa della giusta dimensione***
  - Utilizzare il logaritmo in base 2 (con qualche aggiustamento) per conoscere il numero di cifre necessarie



# Esercizio 1 – main

```
int main()
{
  stack binaryStack = newStack();
  int num;
  char * res;
  printf("Numero intero: ");
  scanf("%d",&num);
  res = convertToBinary(num, &binaryStack);
  printf("RES: %s\n", res);
  return 0;
}
```

# Esercizio 1 – Salvataggio su stringa

```
double mylog(int base, float value)
{
   return log(value) / log(base);
}

int calculateSpace(int num)
{
   if(num == 0) return 2; //1 cifra per lo '0' e il terminatore
   else return mylog(2, num) + 2;
}

void consume(stack* s, char* result)
{
   strcpy(result,"");
   while(!isEmptyStack(*s))
        sprintf(result, "%s%d", result, pop(s) );
}
```



# Esercizio 1 – Funzione di conversione

```
char* convertToBinary(int num, stack* binaryStack)
{
   //allocazione memoria per la stringa risultato
   char* res = (char*) malloc(calculateSpace(num));

   //inserisco il numero di partenza nello stack
   push(num, binaryStack);

   //lancio l'algoritmo
   calculate(binaryStack);

   //salvataggio su stringa
   consume(binaryStack, res);

   return res;
}
```

# Esercizio 1 – Algoritmo

```
void calculate(stack* s)
{
   element bit, curVal;
  curVal = pop(s);
  if(curVal == 0 || curVal == 1)
      push(curVal, s); //calcolo terminato
  else
  {
      bit = curVal % 2;
      curVal = curVal / 2;
      push(bit, s);
      push(curVal, s);
      calculate(s); //rilancia il calcolo
  }
}
```



# Esercizio 2 – Stack

**Riconoscimento palindromi mediante stack**

- Si realizzi un programma che, ***utilizzando (due) stack***, prenda in input una stringa e valuti se essa è palindroma o meno
- Una stringa è palindroma se la seconda metà è esattamente speculare rispetto alla prima metà
- Esempi:
  - “radar” è palindroma
  - “non” è palindroma
  - “nono” non è palindroma
  - “onorarono” è palindroma

# Esercizio 2 – Soluzione (1)

```
Boolean equals(StackElement e1, StackElement e2)
{
    return e1 == e2;
}
Boolean isPalindrome(char* str)
{
    int i;
    Stack s1 = newStack(), s2 = newStack();
    Boolean palindrome = true;
    int length = strlen(str);
    for(i = 0; i < length; i++)
    {
          push(str[i], s1);
    }
    …
```



# Esercizio 2 – Soluzione (2)

```
    …
    for(i = 0; i < length / 2; i++)
    {
          push(pop(s1), s2);
    }
    if(length % 2 == 1)
    {
          pop(s1);
    }
    while(!isEmptyStack(s1) && palindrome)
    {
          palindrome = equals(pop(s1),pop(s2));
    }
    return palindrome;
}
```

# Esercizio 3

## Coda FIFO per la playlist di un jukebox

- `void playSong(startQueue* start, endQueue* end)`
  - Simula la riproduzione della prossima canzone, stampandone il contenuto sullo standard output
- `void printPlaylist(startQueue* start, endQueue* end)`
  - Stampa tutte le canzoni sullo standard output
  - Alla fine della stampa la coda deve tornare allo stato iniziale (si utilizzi una nuova coda accessoria)
- `int insertSong(Song el, startQueue* start, endQueue* end)`
  - Inserisce una nuova canzone, ma solo se non è già presente nella playlist
  - Alla fine del controllo ***la coda deve tornare allo stato iniziale (NON si utilizzi una nuova coda accessoria)***

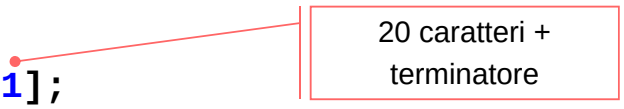


# Esercizio 3 - Strutture dati

Tipi di dato utilizzati

```
typedef struct
{
    char autore[21];
    char canzone[21];
    float durata;
}Song;

typedef Song element;
```



# Esercizio 3 - playSong

```
void playSong(startQueue* start, endQueue* end)
{
   Song el;
   el = deQueue(start,end);
   printf("PLAYSONG: autore %s, canzone %s, duarata %f\n",
                          el.autore,el.canzone,el.durata);
    return;
};
```

Suonare una canzone ne implica anche l'eliminazione dalla playlist



# Esercizio 3 - playSong

```
void printPlaylist(startQueue* start, endQueue* end)
{
    startQueue newStart;      endQueue newEnd;
    Song el;
    createEmptyQueue(&newStart, &newEnd);
    while(!isEmptyQueue(*start))
    {
        el=deQueue(start,end);
        printf("autore %s, canzone %s, duarata %f\n",
                          el.autore,el.canzone,el.durata);
        enQueue(el,&newStart,&newEnd);
    }
    *start=newStart;
    *end=newEnd;
    return;
}
```

Creo una nuova coda

Inserisco nella nuova coda le canzoni eliminate dalla coda data

Sostituisco la coda data con la nuova coda appena creata

# Esercizio 3 - insertSong

```
int insertSong(Song el, startQueue* start, endQueue* end)
{
    if(isEmptyQueue(*start))
    {
        enQueue(el, start, end);
        return 1;
    }
    else
        if( cercaElemento(el, start, end)==0 )
        {
            enQueue(el, start, end);
            return 1;
        }
        else return 0;
}
```

Coda con un unico elemento

Funzione di supporto che restituisce 0 se l'elemento dato esiste



# Esercizio 3 - insertSong

```
int cercaElemento(Song el, startQueue* s, endQueue* e){
    Song temp;    int trovato = 0, last=0;
    endQueue endIniziale = *e;
    do
    {
        if(endIniziale==*s)
            last=1;
        temp=deQueue(s, e);
        if( (strcmp(temp.autore,el.autore)==0)
                && (strcmp(temp.canzone,el.canzone)==0) )
                    trovato = 1;
        enQueue(temp,s,e);
    }
    while(last==0);
    return trovato;
}
```

Memorizzo l'indirizzo dell'ultimo elemento della coda iniziale

L'elemento attualmente in considerazione ha lo stesso indirizzo dell'ultimo elemento della coda iniziale

L'elemento appena eliminato dalla testa della coda viene inserito alla fine della coda

Il ciclo termina solo quando la coda iniziale è stata ricostruita completamente