La memoria si divide in due “blocchi” principali: STACK e HEAP.

Guardiamo il seguente codice:

#include <stdio.h>

#include <stdlib.h>

int quadrato (int y);

int cubo (int y);

int main (int argc, char \*argv[]){

int x;

int Q[11];

int C[11];

for (x = 1; x <= 10; x++){

Q[x] = quadrato(x);

C[x] = cubo(x);

}

return EXITSUCCESS;

}

int quadrato (int y){

return y \* y;

}

int cubo (int y){

int q;

q = quadrato(y);

return q \* y;

}

Qui, all’esecuzione abbiamo il seguente schema di memoria:

STACK----------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

x[ ]

Q[ ]

C[ ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I valori di argc e argv sono forniti dal sistema operativo all’eseguibile al momento dell’esecuzione

Supponiamo di aver già fatto i conti per 1 e 2, e di avere X=3. Al momento della chiamata a Cubo accadrà questo nella memoria:

STACK----------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

x[3]

Q[1,4,9 ]

C[1,8 ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cubo\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

y [3]

q [ ]

-- [ ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*Quadrato\*\*\*\*\*\*\*\*\*\*\*\*\*

y [3]

-- [9]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Questo è lo stato della memoria al momento della chiamata ed esecuzione di quadrato, subito dopo la memoria di quella sottofunzione viene deallocata e il valore di output è salvato in q nella memoria di cubo:

STACK----------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

x[3]

Q[1,4,9 ]

C[1,8 ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cubo\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

y [3]

q [9]

-- [27]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Subito dopo il valore viene salvato e la memoria di Cubo viene deallocata:

STACK----------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

x[3]

Q[1,4,9 ]

C[1,8,27 ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Eccetera.

Allocazione dinamica:

Se nel mio codice ho un’allocazione del tipo

int main( int argc, int \*argv[]){

int \*P;

int n;

%robachelegge n%

P=(int \*) malloc(n\*sizeof(int))

}

Nella mia memoria avrò qualcosa del tipo:

STACK---------------------------- HEAP-----------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

n[numeroIntero]

P--------------------------------------------🡪 [ ] //n celle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

E come si comporta la memoria quando lo faccio in una procedura?

int main( int argc, int \*argv[]){

int \*P;

int n;

%robachelegge n%

P=alloca(n);

}

Int \*alloca (int n){

Return (int \*) malloc(n\*sizeof(int))

}

Con codice simile la memoria sarà simile alla seguente:

STACK---------------------------- HEAP-----------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

n[numeroIntero]

P[ ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*alloca\*\*\*\*\*\*\*\*\*\*\*\*\*\*

n[ ]

-- [ ]---------------------------------------🡪 [ ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Dopo l’esecuzione della procedura avviene la deallocazione della memoria della procedura e il passaggio del risultato, che è un puntatore:

STACK---------------------------- HEAP-----------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

argc[1]

argv[“NomeEseguibile”]

--

n[numeroIntero]

P[ ]------------------------------------+

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

|

|

|

+----🡪 [ ]