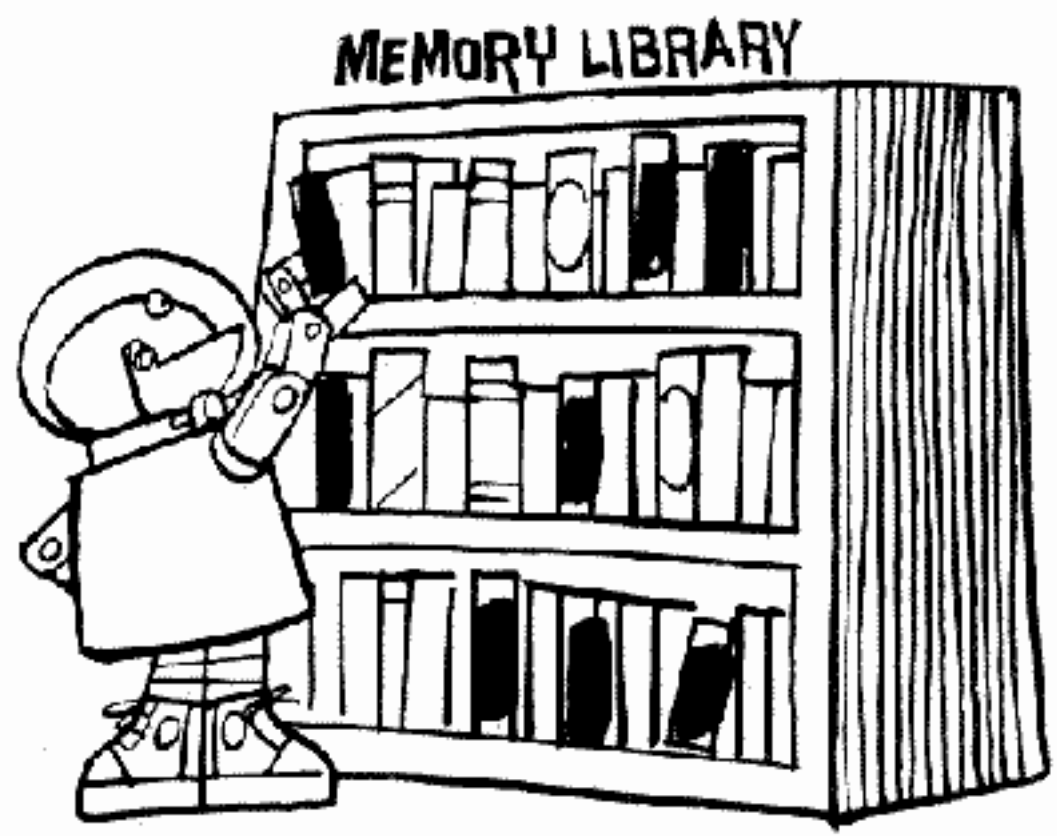
CSE 251 - pointer

Arend Hintze

rememer memory?

- the computer is super picky
- everything has to be in its place
- never put something else than the toothbrush where toothbrushes belong!



variables need:

- needs a name
- has a type
- has a place
- is filled with content

variables need:

- needs a name
- has a type
- has a place
- is filled with content

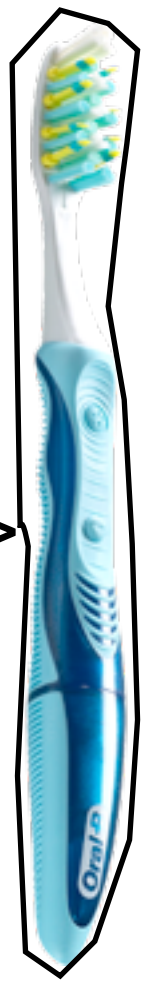
myToothbrush

variables need:

- needs a name myToothbrush
- has a type it's a toothbrush - for crying out loud!
- has a place
- is filled with content

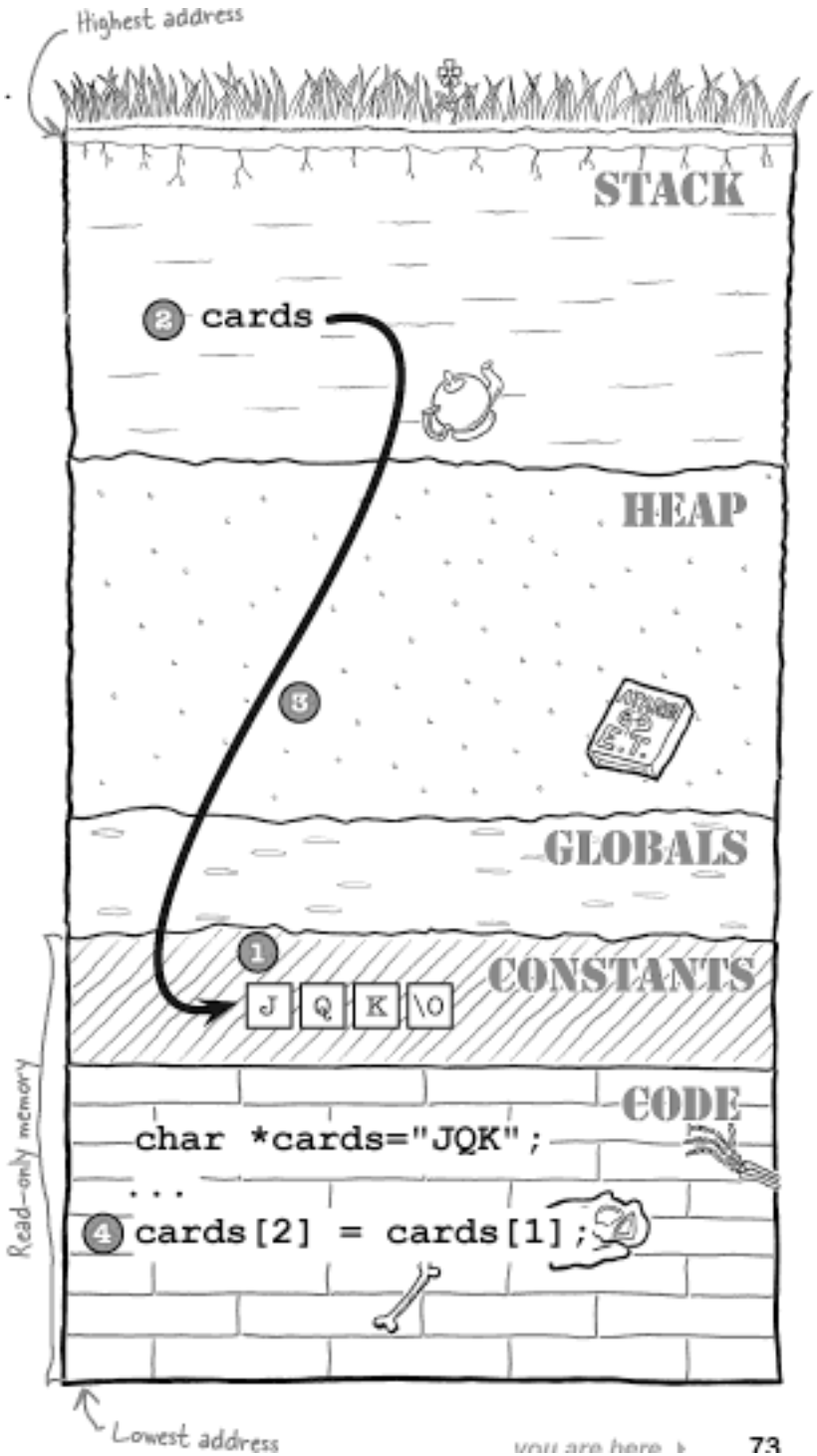
variables need:

- needs a name myToothbrush
- has a type it's a toothbrush - for crying out loud!
- has a place here ->
- is filled with content

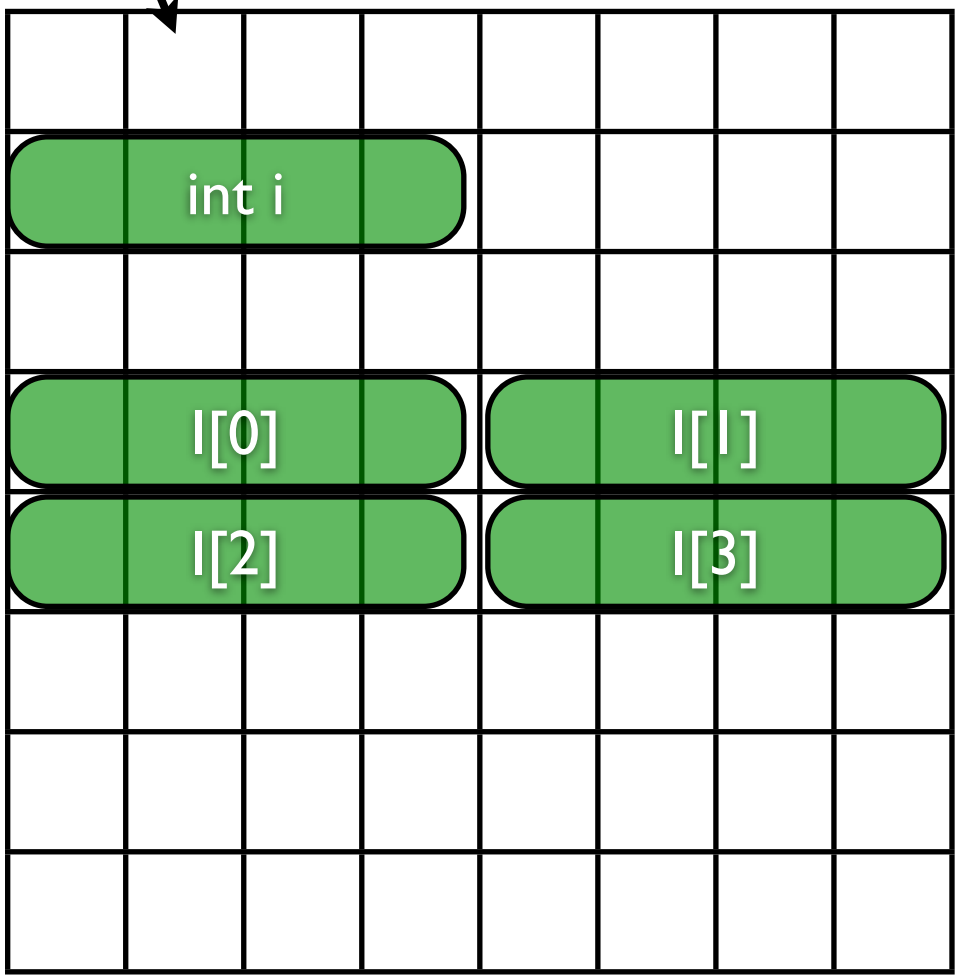


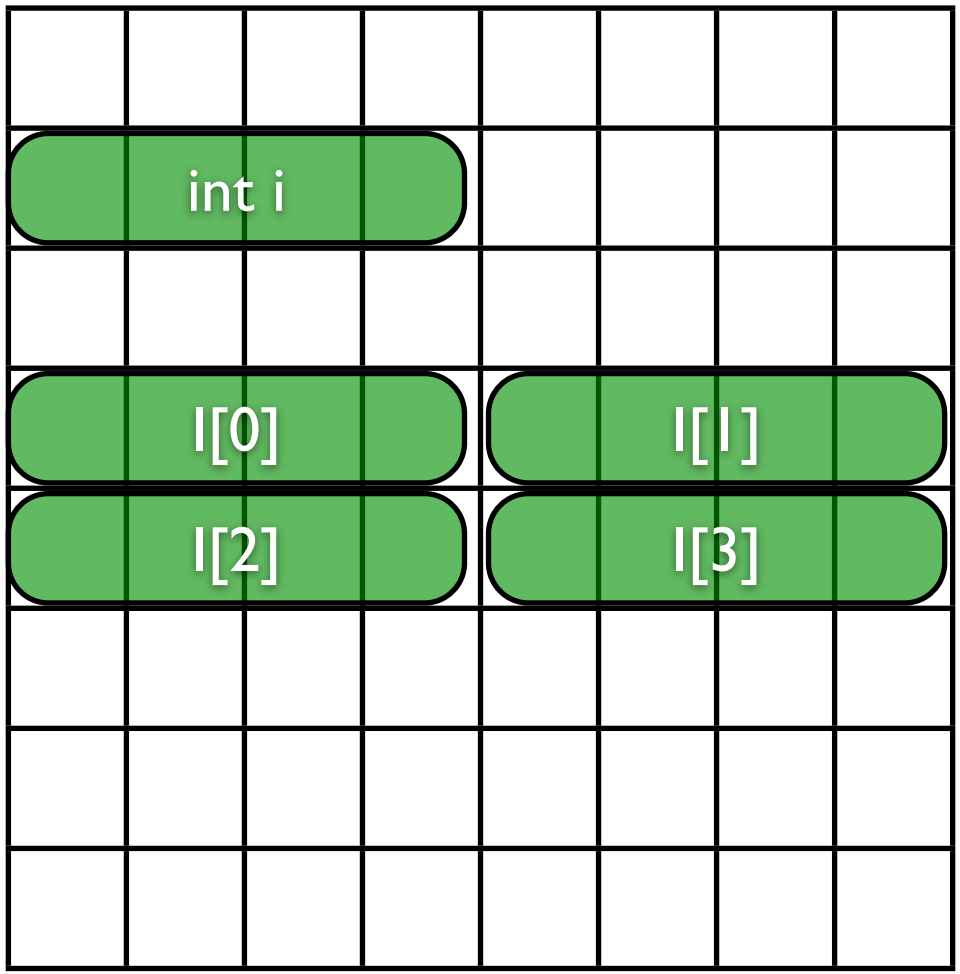
variables need:

- needs a name — myToothbrush
- has a type — it's a toothbrush - for crying out loud!
- has a place — here ->
- is filled with content — the real thing! ->

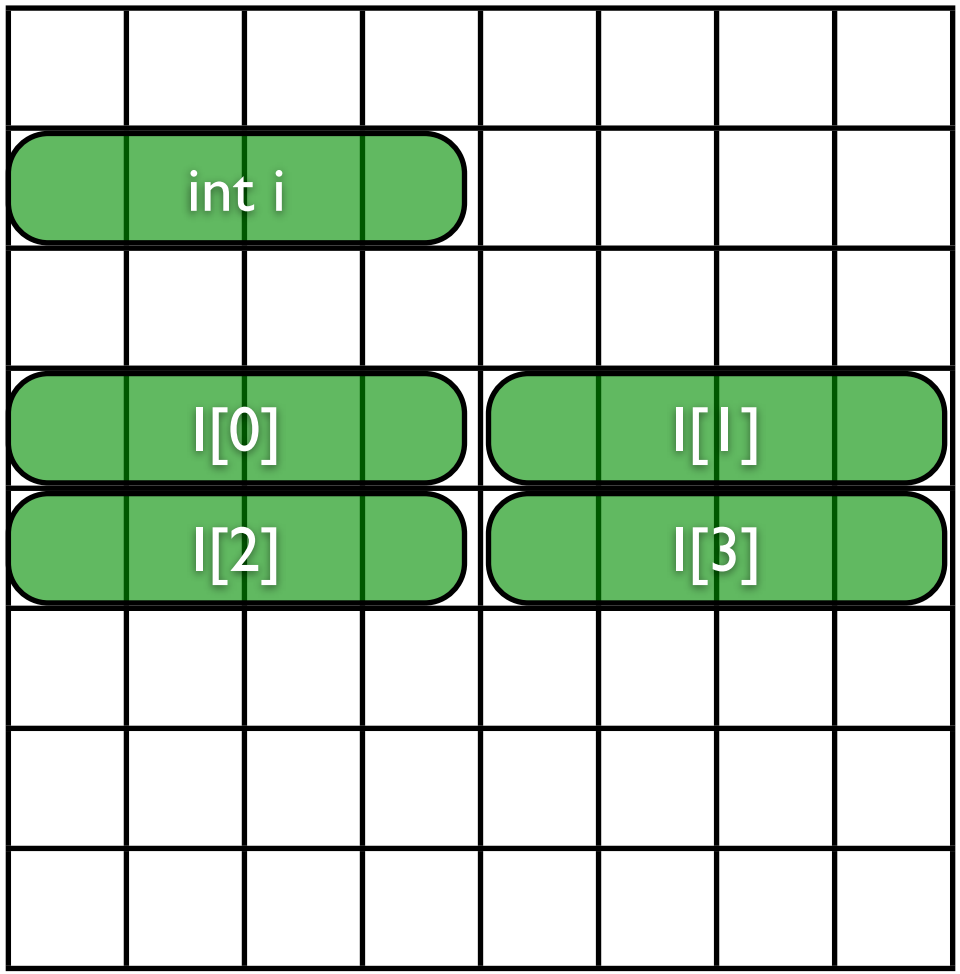

Highest address
STACK
2 cards
HEAP
3
GLOBALS
1
J Q K \0
CONSTANTS
CODE
char *cards="JQK";
...
4 cards[2] = cards[1];
Read-only memory
Lowest address

byte = 8 bits


int i
I[0]
I[1]
I[2]
I[3]

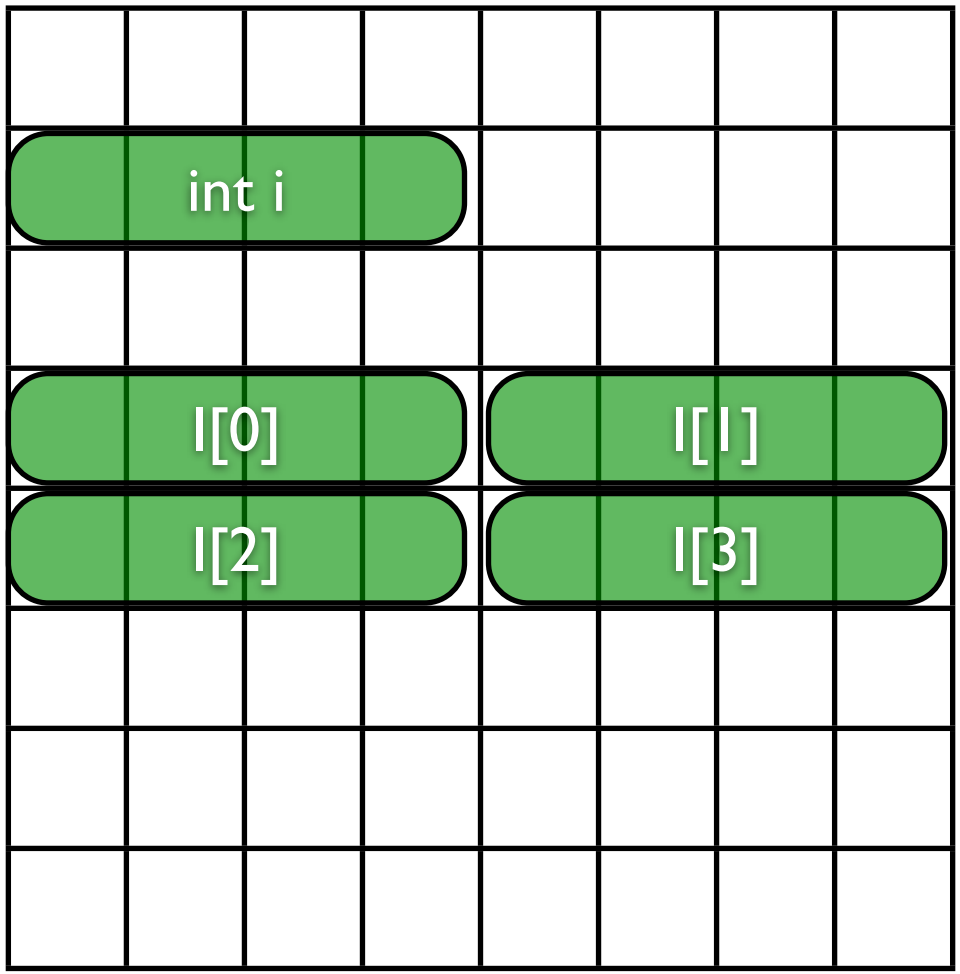



```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```




```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

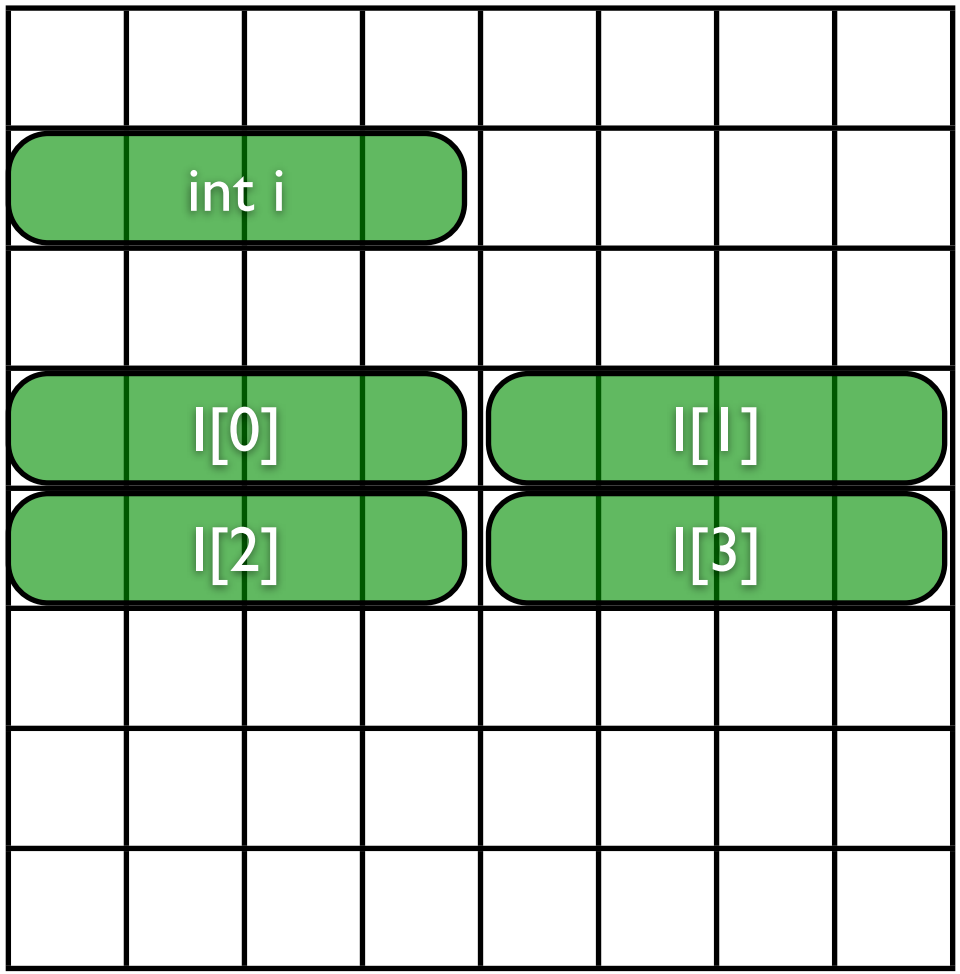
0 0 0




```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

0 0 0

```
printf("%i  %i  %i\n",I,&I,&I[0]);
```

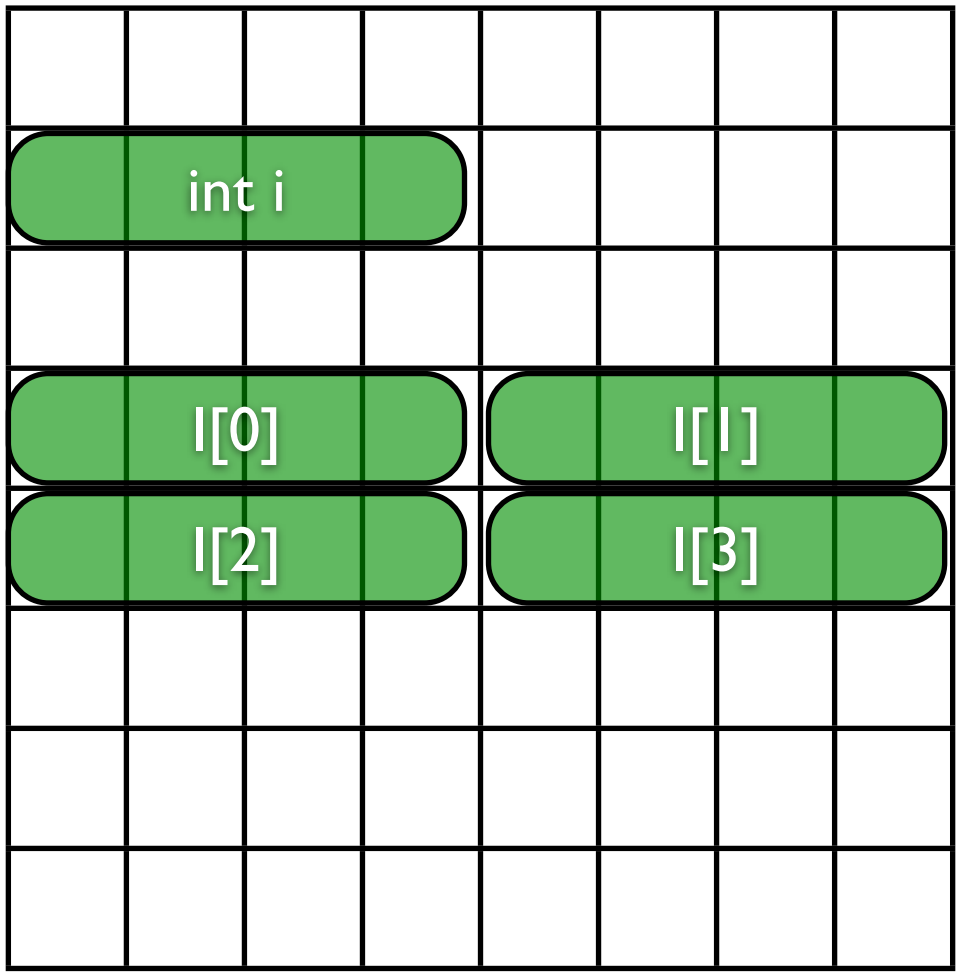



```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

0 0 0

```
printf("%i  %i  %i\n",I,&I,&I[0]);
```

1606416784 1606416784 1606416784



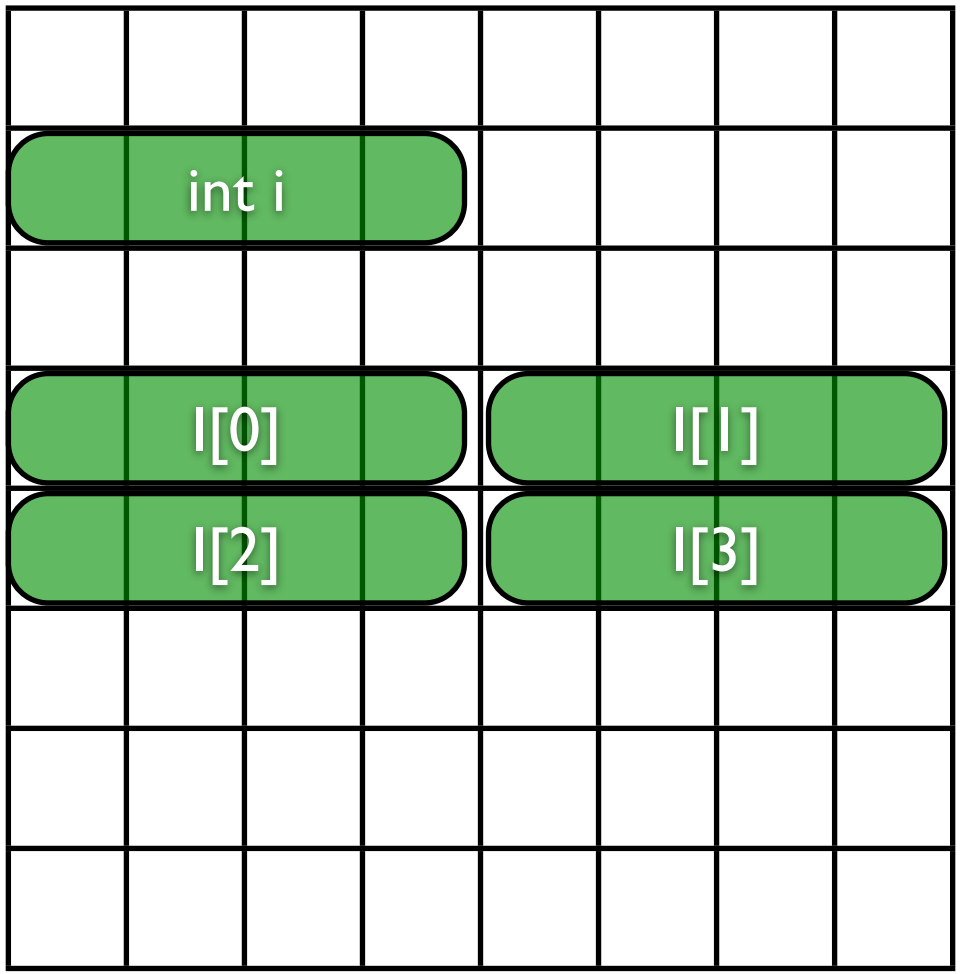

```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

0 0 0

```
printf("%i  %i  %i\n",I,&I,&I[0]);
```

1606416784 1606416784 1606416784

I[0] = name I[0], content 0, type int, address




```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

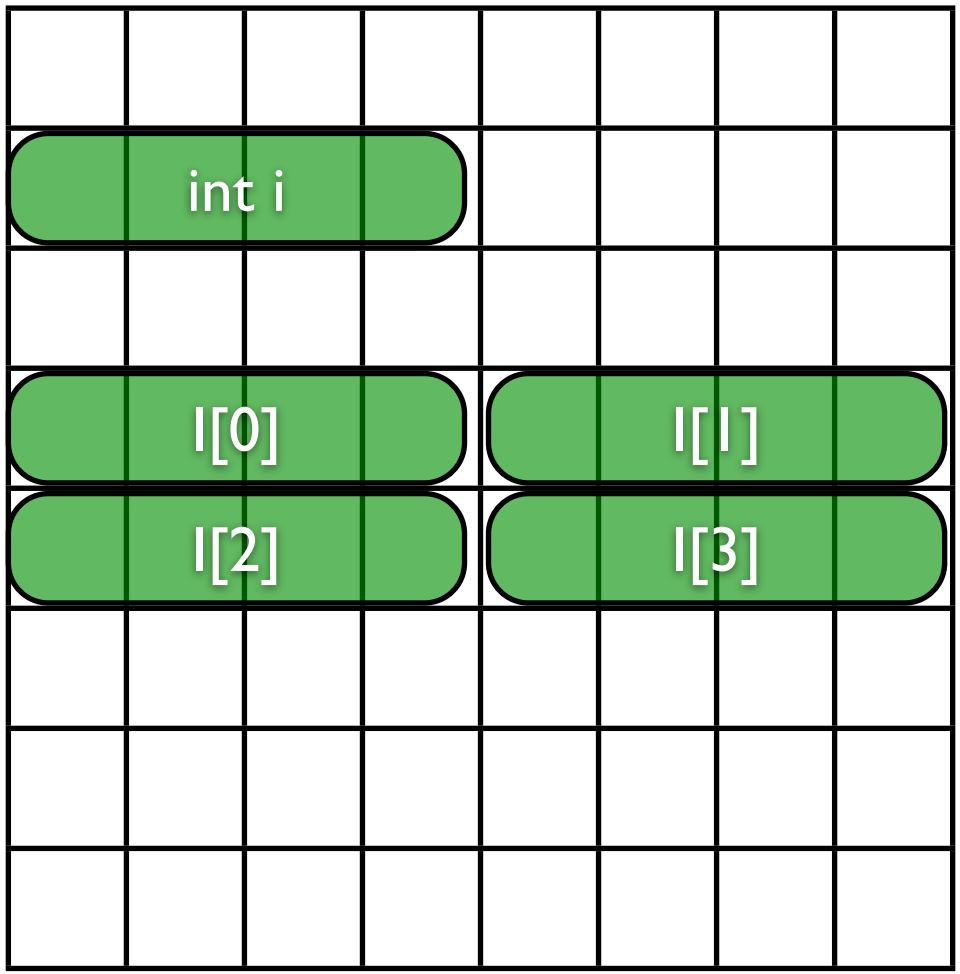
0 0 0

```
printf("%i  %i  %i\n",I,&I,&I[0]);
```

```
1606416784  1606416784  1606416784
```

I[0] = name I[0], content 0, type int, address

I = name I, content 1606416784, type int[], 1606416784




```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

0 0 0

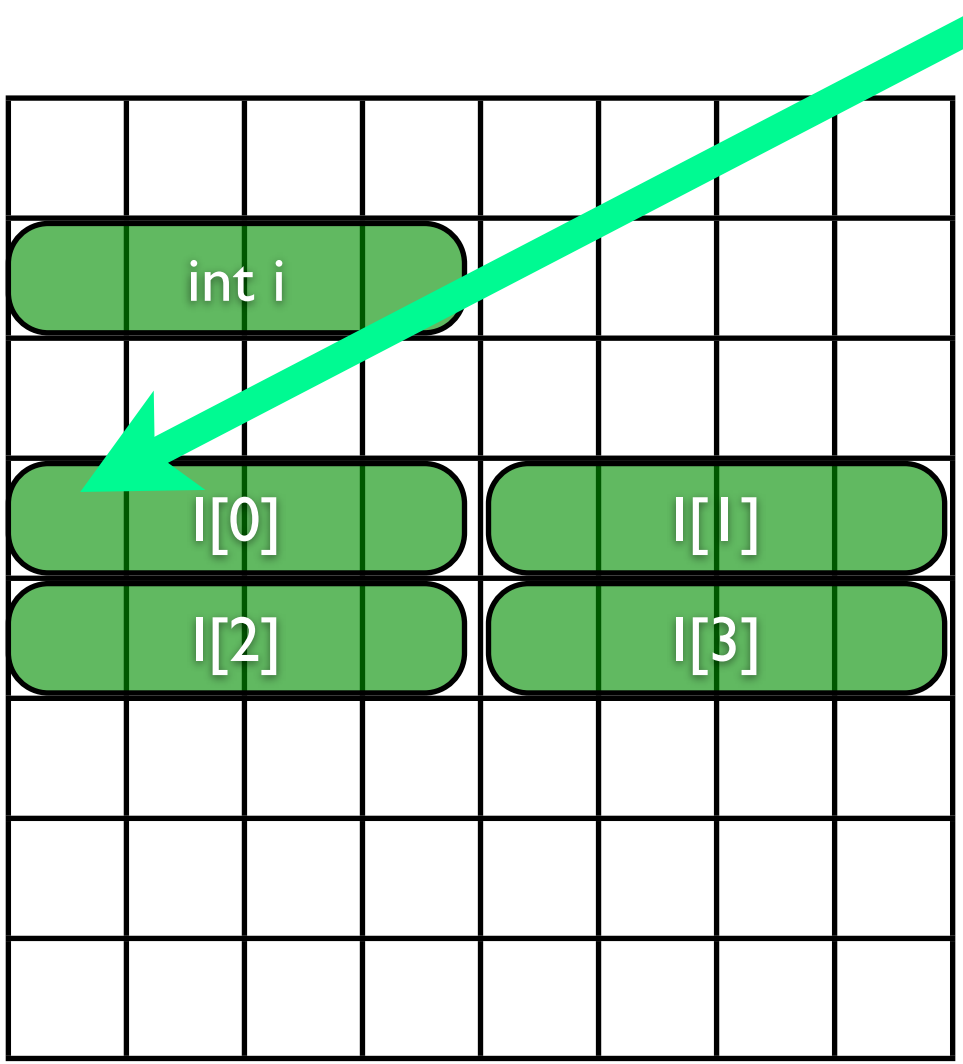
```
printf("%i  %i  %i\n",I,&I,&I[0]);
```

```
1606416784  1606416784  1606416784
```

I stores the address where the array begins!

I[0] = name I[0], content 0, type int, address

I = name I, content 1606416784, type int[], 1606416784




```
int i=0;
int I[4]={0,1,2,3};
printf("%i %i %i\n",i,I[0],I[i]);
```

0 0 0

```
printf("%i  %i  %i\n",I,&I,&I[0]);
```

```
1606416784  1606416784  1606416784
```

I stores the address where the array begins!

I[0] = name I[0], content 0, type int, address

I = name I, content 1606416784, type int[], 1606416784

a new "type": pointer

- every classical type can have a pointer to it
- the pointer is indicated by a *
- int *i is a pointer to an int
- pointers are not initialized!
- the memory address is stored in i
- the value of its memory address is in *i

```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

```
42 1606416808 1606416800 1606416808 42
```

```c
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

```
 42 1606416808 1606416800 1606416808 42
```

```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

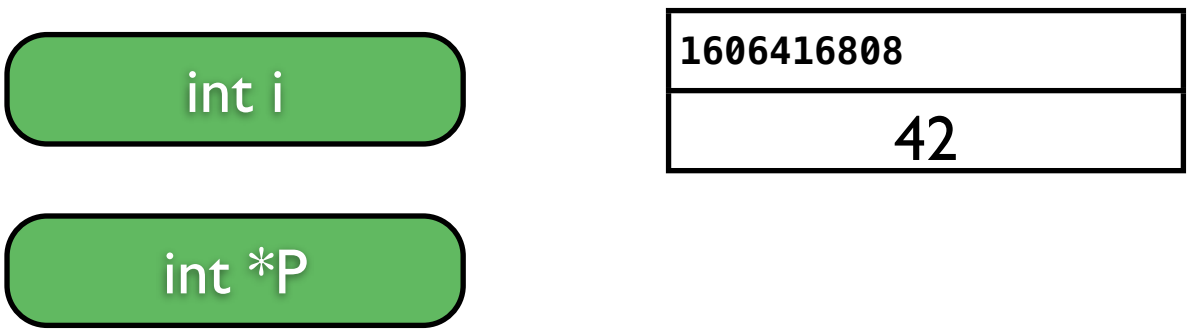
```
42 1606416808 1606416800 1606416808 42
```

int i

int *P

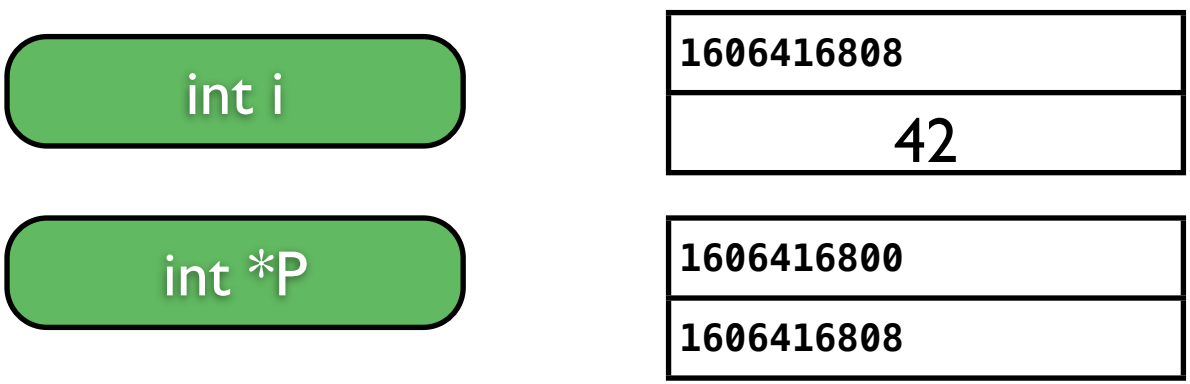
```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

42 1606416808 1606416800 1606416808 42



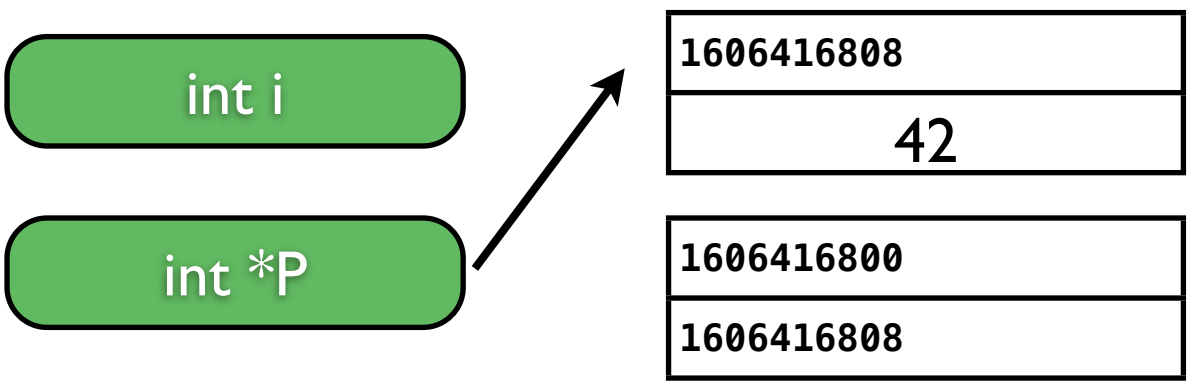
```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

42 1606416808 1606416800 1606416808 42



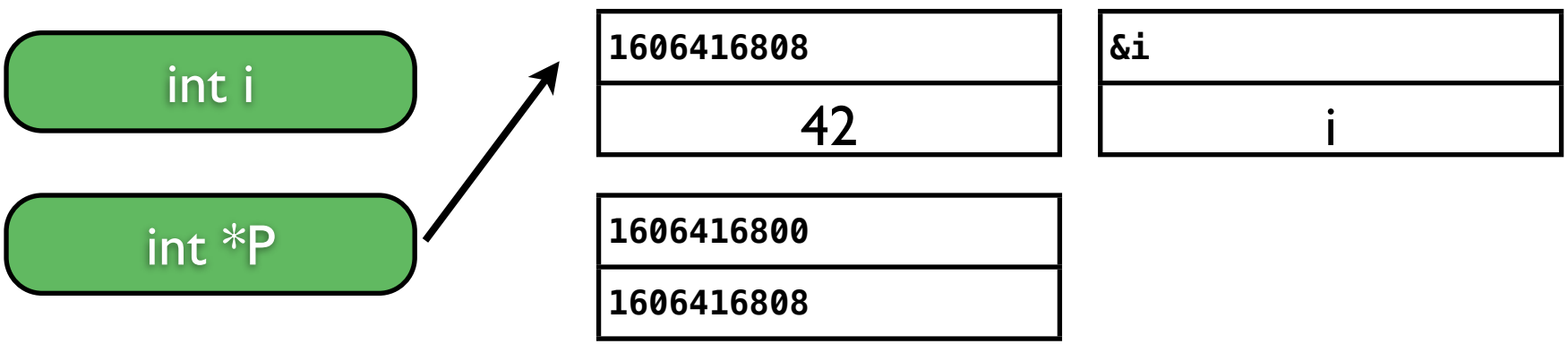
```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

42 1606416808 1606416800 1606416808 42



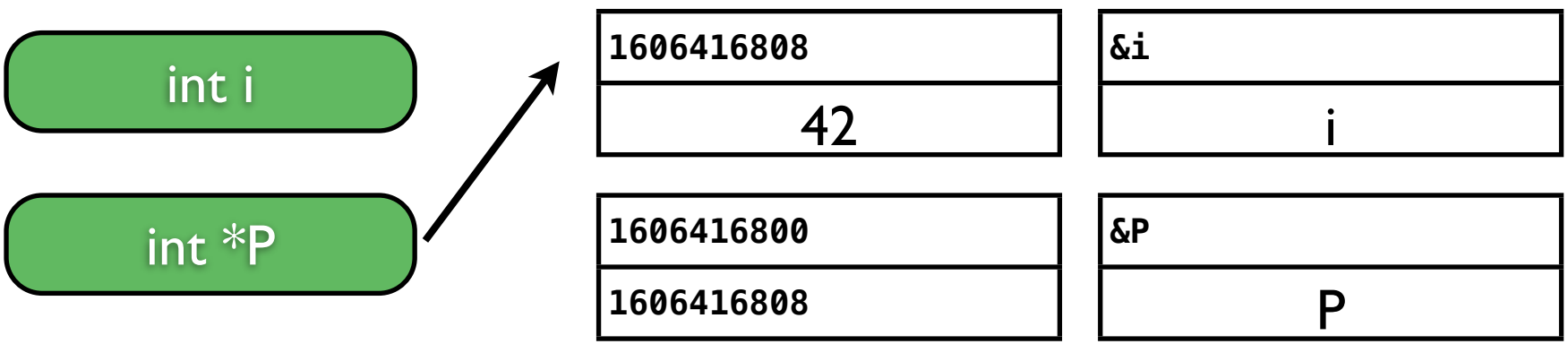
```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

42 1606416808 1606416800 1606416808 42



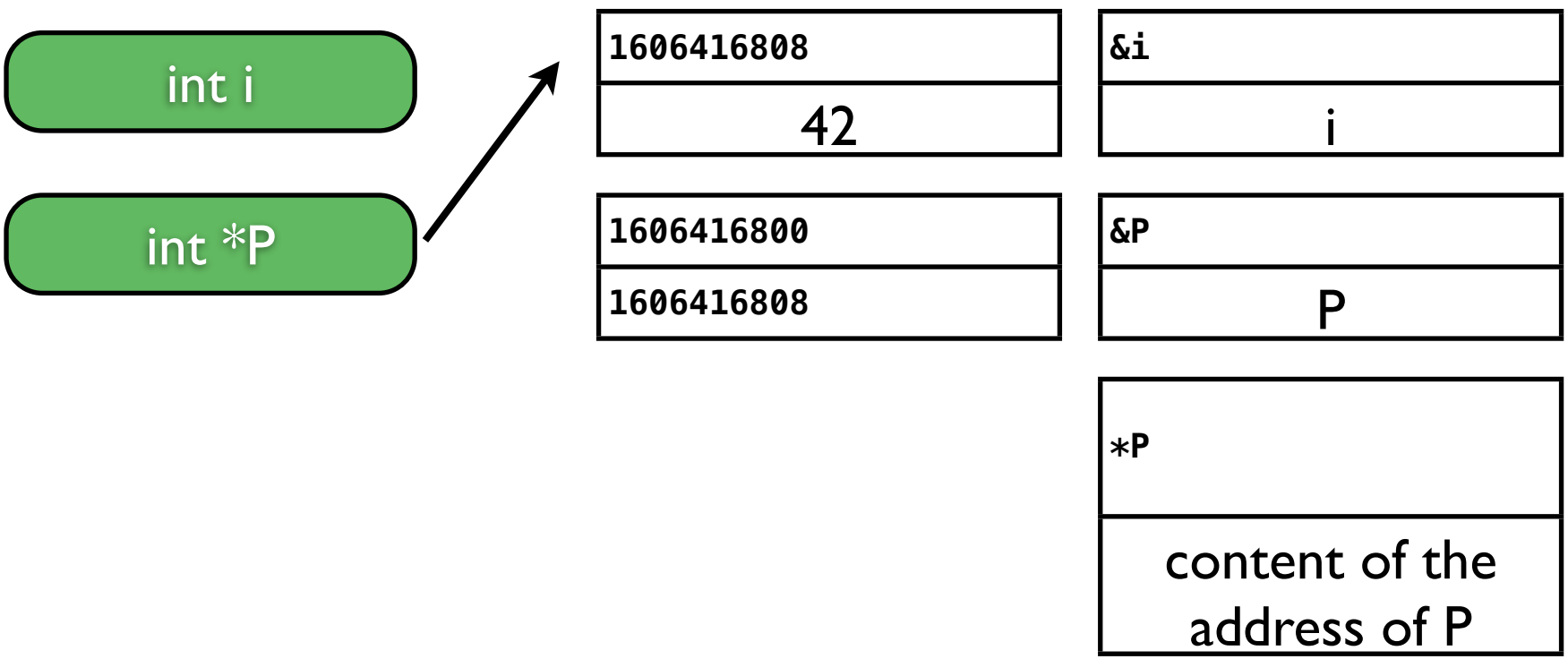
```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

42 1606416808 1606416800 1606416808 42



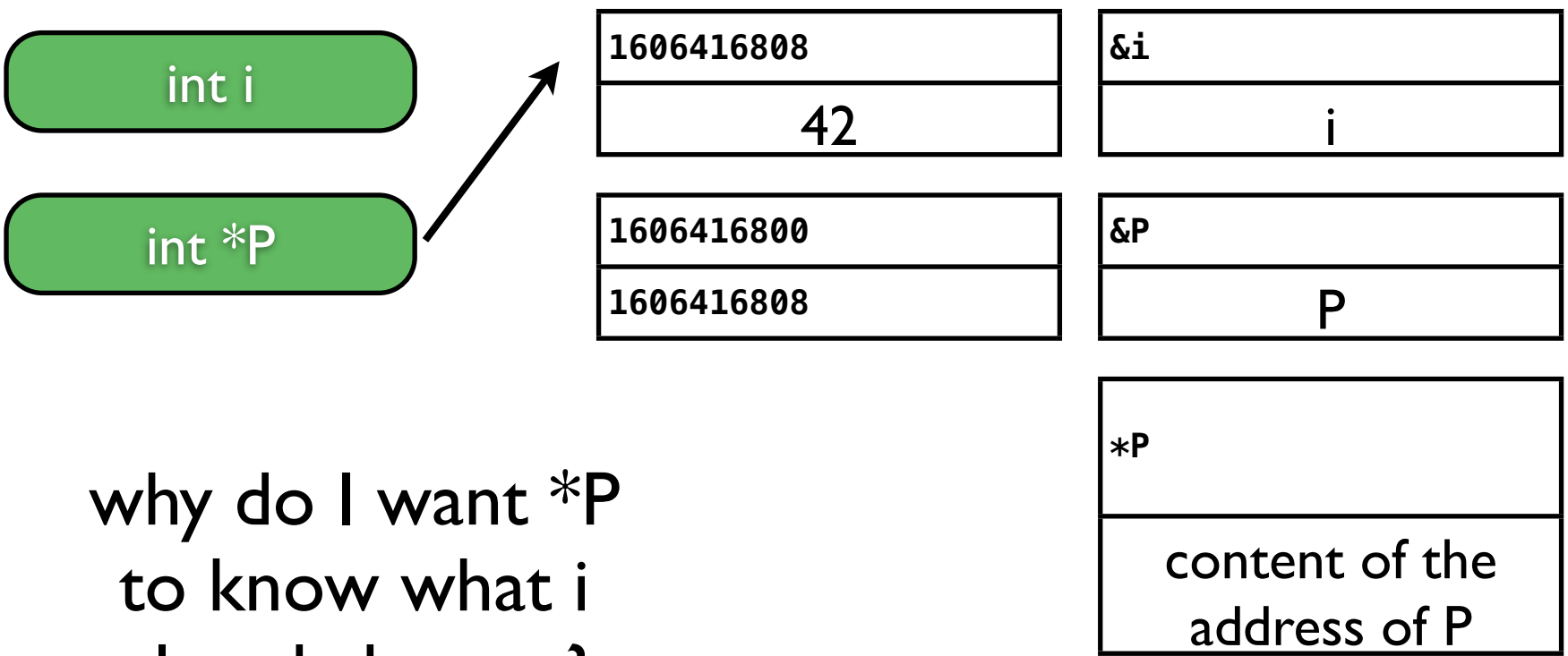
```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

42 1606416808 1606416800 1606416808 42



```
int i=42;
int *P=&i;
printf("%i %i %i %i %i\n",i,&i,&P,P,*P);
```

```
42 1606416808 1606416800 1606416808 42
```



why do I want *P to know what i already knows?

malloc, sizeof, and free

- instead of pointing to something already existing, you can ask for memory
- malloc gives you bytes
- sizeof tells you how many bytes in a type
- free releases this memory
- [n] is an offset to a pointer

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

int main(){
    int i;
    int *I;
    I=malloc(sizeof(int)*4);
    for(i=0;i<4;i++)
        I[i]=rand();
    for(i=0;i<4;i++)
        printf("%i %i\n",i,I[i]);;
    free(I);
    return 0;
}
```

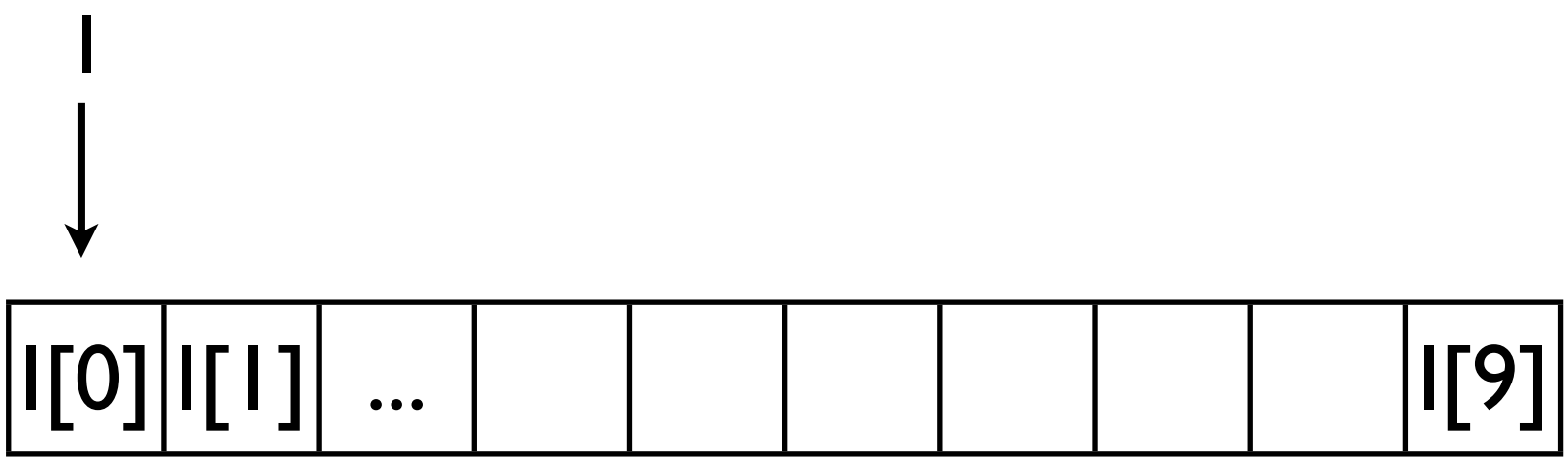
```c
better:
I=(int*)malloc(sizeof(int)*4);
```

```
int I[10];
```

synonymous to

```
int *I=(int*)malloc(sizeof(int)*10);
```

```
if(*I==I[0]){
    printf("allways true!\n");
}
```



todo

- make a random number between 100 and 200 called N
- get memory to store all N elements
- make N random numbers stor them in the new found memory
- sum all emements in this array, print the result

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

int sum(int *A,int N);

int main(){
    int i;
    int *I;
    I=(int*)malloc(sizeof(int)*4);
    for(i=0;i<4;i++)
        I[i]=rand()&255;
    for(i=0;i<4;i++)
        printf("%i %i\n",i,I[i]);
    printf("sum %i\n",sum(I,4));
    free(I);
    return 0;
}
```

```c
int sum(int *A,int N){
    int i,s=0;
    for(i=0;i<N;i++)
        s+=A[i];
    return s;
}
```

using pointers in functions

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

int* makeNRandomNumbers(int N);
int sum(int *A,int N);
void showNNumbers(int *A,int N);

int main(){
    int i;
    int *I;
    I=makeNRandomNumbers(100);
    showNNumbers(I, 100);
    printf("sum %i\n",sum(I,100));
    free(I);
     return 0;
}
```

```c
int sum(int *A,int N){
    int i,s=0;
    for(i=0;i<N;i++)
        s+=A[i];
    return s;
}

int* makeNRandomNumbers(int N){
    int i;
    int *A=(int*)malloc(sizeof(int)*N);
    for(i=0;i<N;i++)
        A[i]=rand()&255;
    return A;
}

void showNNumbers(int *A,int N){
    int i;
    for(i=0;i<N;i++)
        printf("%i %i\n",i,A[i]);
}
```

remember this:

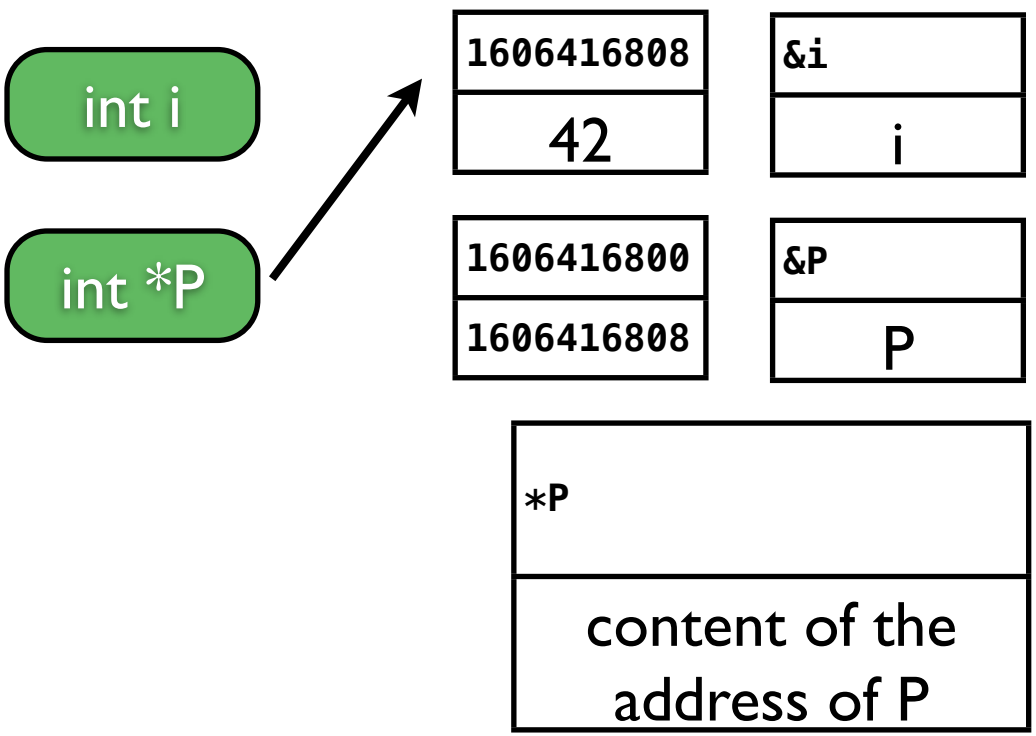
```c
#include <stdio.h>
#include <stdlib.h>

void show(int A){
    printf("%i\n",A);
}

void blank(int A){
    A=0;
    printf("%i\n",A);
}

int main(){
    int i=5;
    show(i);
    blank(i);
    show(i);
    return 0;
}
```

call by reference I

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

int* makeNRandomNumbers(int N);
int sum(int *A,int N);
void showNNumbers(int *A,int N);
void zeroAllElements(int *A,int N);

int main(){
    int i;
    int *I;
    I=makeNRandomNumbers(10);
    showNNumbers(I, 10);
    zeroAllElements(I, 10);
    showNNumbers(I,10);
    printf("sum %i\n",sum(I,10));
    free(I);
     return 0;
}

int sum(int *A,int N){
    int i,s=0;
    for(i=0;i<N;i++)
        s+=A[i];
    return s;
}
```

```c
int* makeNRandomNumbers(int N){
    int i;
    int *A=(int*)malloc(sizeof(int)*N);
    for(i=0;i<N;i++)
        A[i]=rand()&255;
    return A;
}

void showNNumbers(int *A,int N){
    int i;
    for(i=0;i<N;i++)
        printf("%i %i\n",i,A[i]);
}
void zeroAllElements(int *A,int N){
    int i;
    for(i=0;i<N;i++)
        A[i]=0;
}
```

call by reference II



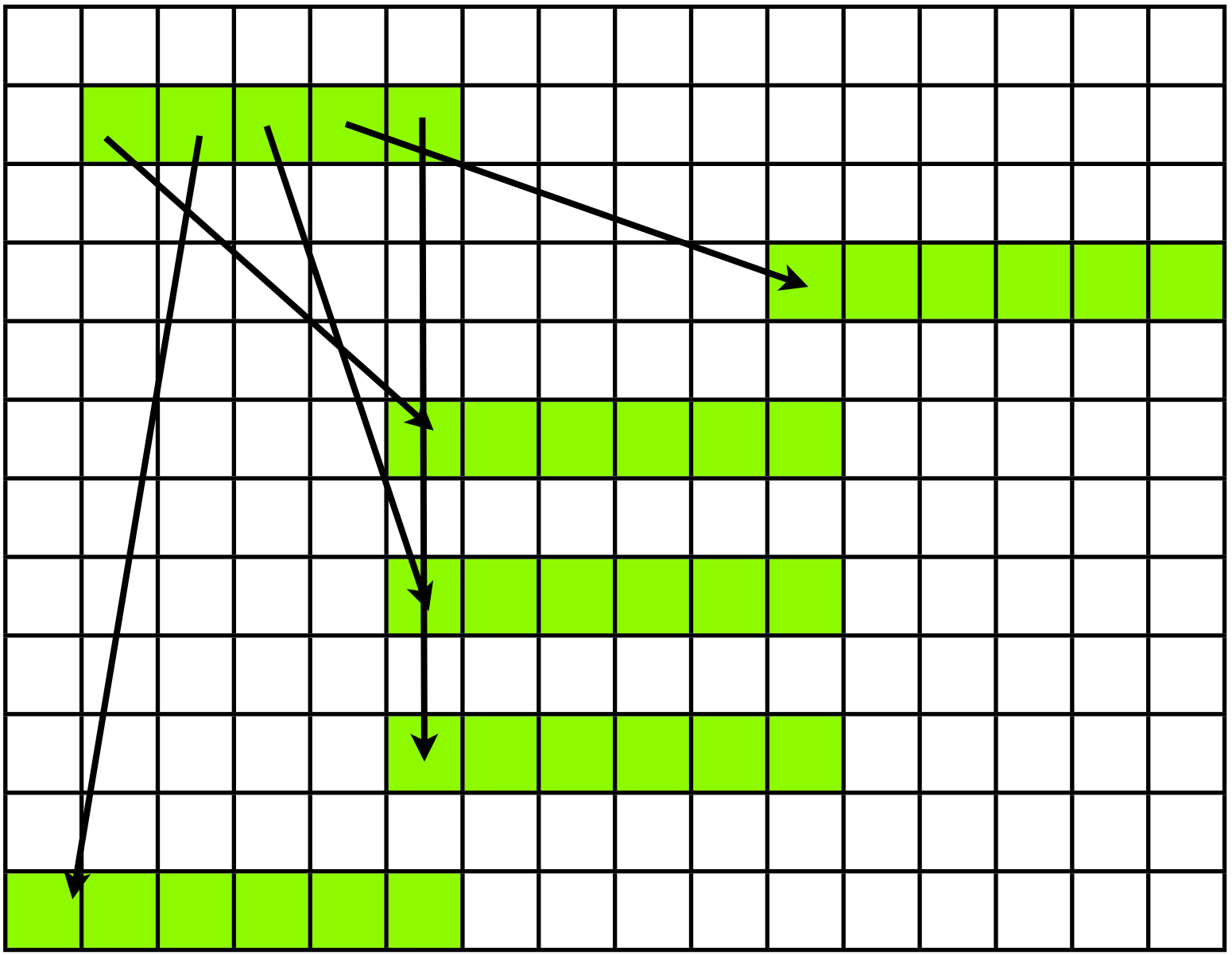

```c
#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int*b);

int main(){
    int a=6,b=7;
    printf("%i %i\n",a,b);
    swap(&a,&b);
    printf("%i %i\n",a,b);
    return 0;
}

void swap(int *a, int*b){
    int d=*a;
    *a=*b;
    *b=d;
}
```

recap pointers here:

- http://www.thegeekstuff.com/2011/12/c-pointers-fundamentals/
- http://www.thegeekstuff.com/2012/01/advanced-c-pointers/
- http://www.learnconline.com/2010/03/pointers.html
- http://cplus.about.com/od/learningc/ss/pointers.htm
- http://www.codingunit.com/c-tutorial-how-to-use-pointers
- http://www.tutorialspoint.com/ansi_c/c_pointing_data.htm
- http://www.tenouk.com/Module8b.html

dynamic 2D arrays

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

int main(){
    int i,j;
    int **I;
    I=(int**)malloc(sizeof(int*)*10);
    for(i=0;i<10;i++){
        I[i]=(int*)malloc(sizeof(int)*10);
        for(j=0;j<10;j++)
            I[i][j]=rand()&15;
    }
    for(i=0;i<10;i++){
        for(j=0;j<10;j++)
            printf("%i ",I[i][j]);
        printf("\n");
    }
    return 0;
}
```



make a datastructure that can take Floyds triangle

- array of pointers to int arrays
- each array is on element bigger
- fill it with floys numbers
- free every thing

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

int main(){
    int i,j,k=0;
    int **I;
    I=(int**)malloc(sizeof(int*)*10);
    for(i=0;i<10;i++){
        I[i]=(int*)malloc(sizeof(int)*(i+1));
        for(j=0;j<i+1;j++)
            I[i][j]=k++;
    }
    for(i=0;i<10;i++){
        for(j=0;j<i+1;j++)
            printf("%i ",I[i][j]);
        printf("\n");
    }
    for(i=0;i<10;i++)
        free(I[i]);
    free(I);
    return 0;
}
```

I am so sorry but functions have pointers too

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

void FAB(int a,int b);
void FBA(int a,int b);

int main(){
    void (*F)(int,int);
    F=&FAB;
    F(1,2);
    F=&FBA;
    F(2,1);
    return 0;
}
```

```c
void FAB(int a,int b){
    printf("AB %i %i\n",a,b);
}
void FBA(int a,int b){
    printf("BA %i %i\n",b,a);

}
```

qsort example

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

int mySortEvaluatorLTH(const void *A,const void *B){
    const int *a=(const int *)A;
    const int *b=(const int *)B;
    return *a-*b;
}
int mySortEvaluatorHTL(const void *A,const void *B){
    const int *a=(const int *)A;
    const int *b=(const int *)B;
    return *b-*a;
}
int main(){
    int i;
    int *I;
    I=(int*)malloc(sizeof(int)*10);
    for(i=0;i<10;i++){
        I[i]=rand()&31;
        printf("%i ",I[i]);
    }
    printf("\n");
    qsort(I,10,sizeof(int),mySortEvaluatorLTH);
    for(i=0;i<10;i++)
        printf("%i ",I[i]);
    printf("\n");
    qsort(I,10,sizeof(int),mySortEvaluatorHTL);
    for(i=0;i<10;i++)
        printf("%i ",I[i]);
    printf("\n");
     return 0;
}
```

homework

- make a program that uses at least one pointer
- let the program printf what it does
- make meaningfull output that illustrates what is going on
- you choose the functionality of the program yourself