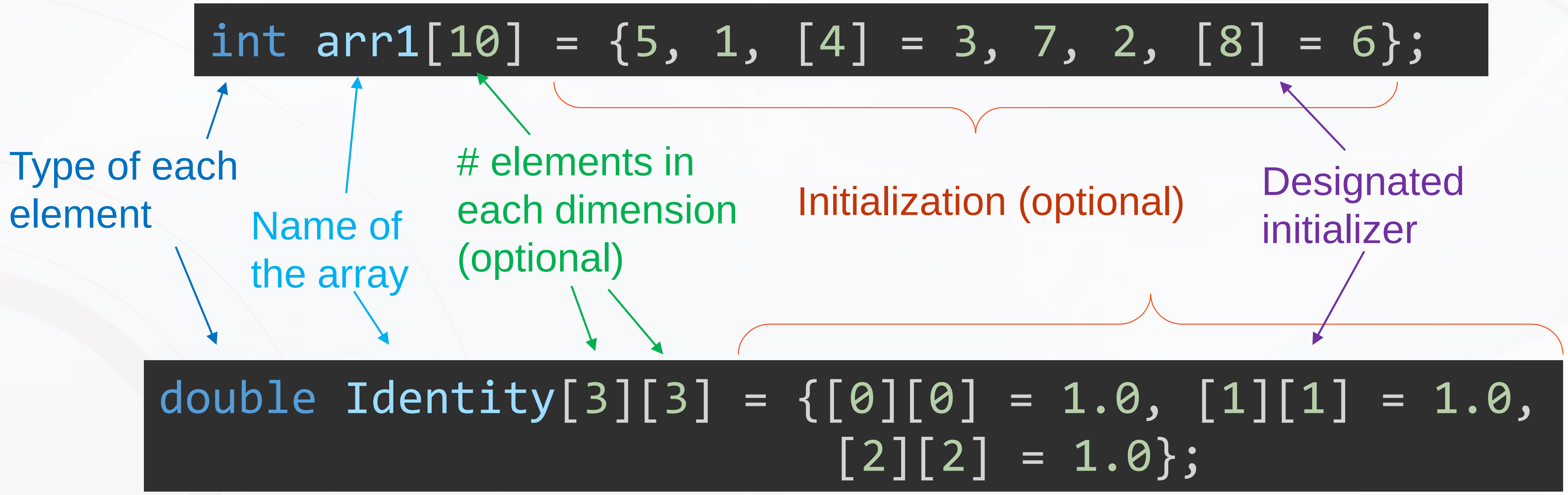
## What have we learned?

### Arrays

- Declaration & initialization



```
int arr1[10] = {5, 1, [4] = 3, 7, 2, [8] = 6};
```

```
double Identity[3][3] = {[0][0] = 1.0, [1][1] = 1.0,
                         [2][2] = 1.0};
```

Type of each element
Name of the array
# elements in each dimension (optional)
Initialization (optional)
Designated initializer

## What have we learned?

- Indexing *Index starts from 0 in C!*

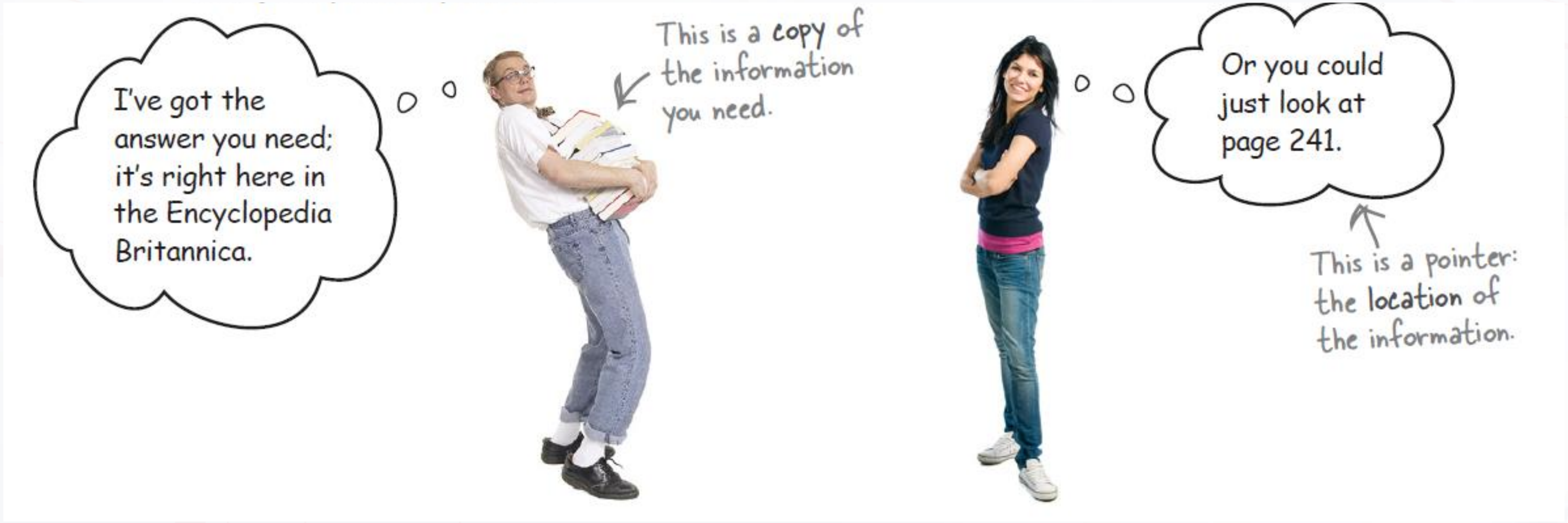
```
#define N 10

    double Identity[N][N];
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (i == j)
                ident[i][j] = 1.0;
            else
                ident[i][j] = 0.0;
        }
    }
```

# Lecture 5　Pointers

## Reasons to use pointers

**Pointer**: Location of a piece of data in memory

- Pass a pointer to avoid passing a whole copy of (a large amount of) data
- Different codes to work on the same piece of data

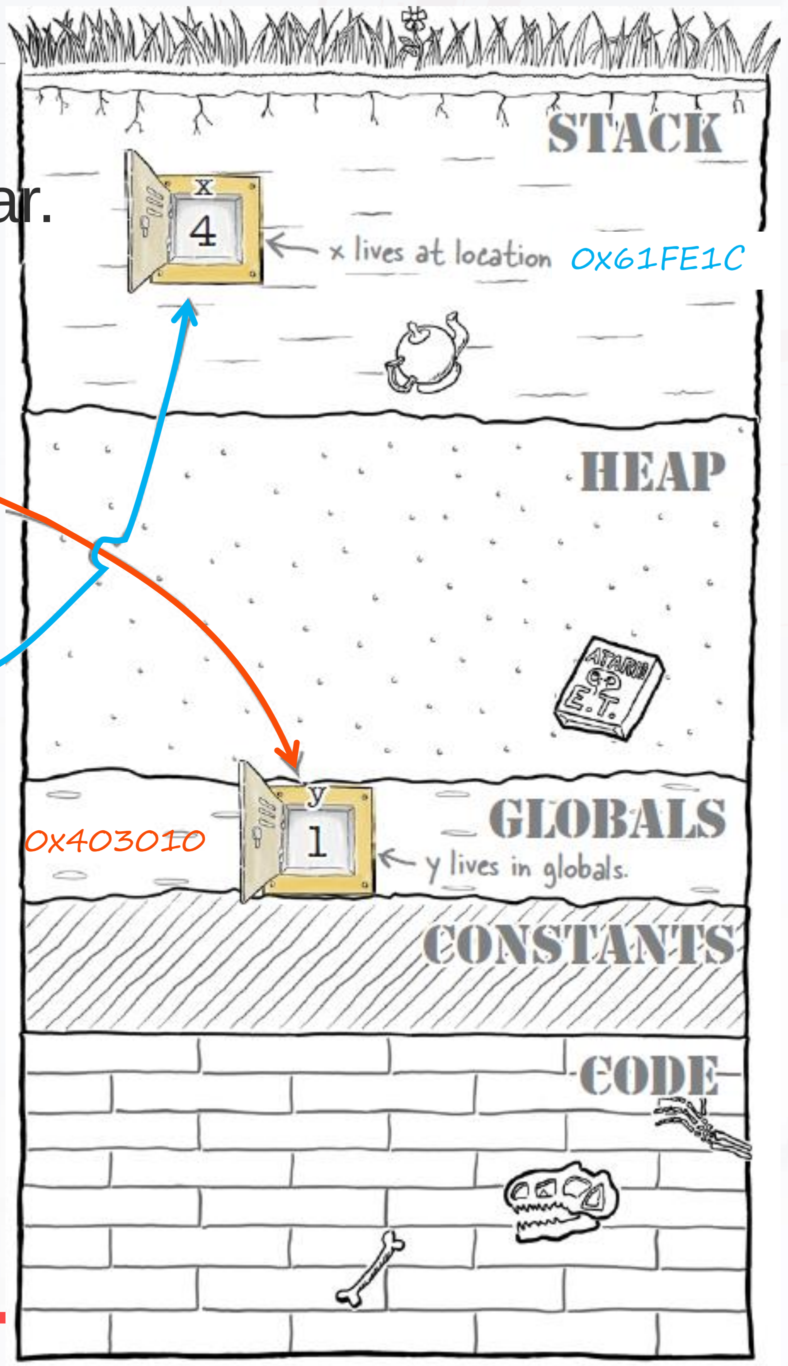

I've got the answer you need; it's right here in the Encyclopedia Britannica.
This is a copy of the information you need.
Or you could just look at page 241.
This is a pointer: the location of the information.

## The & operator

&var – to obtain the memory address (pointer) of var.

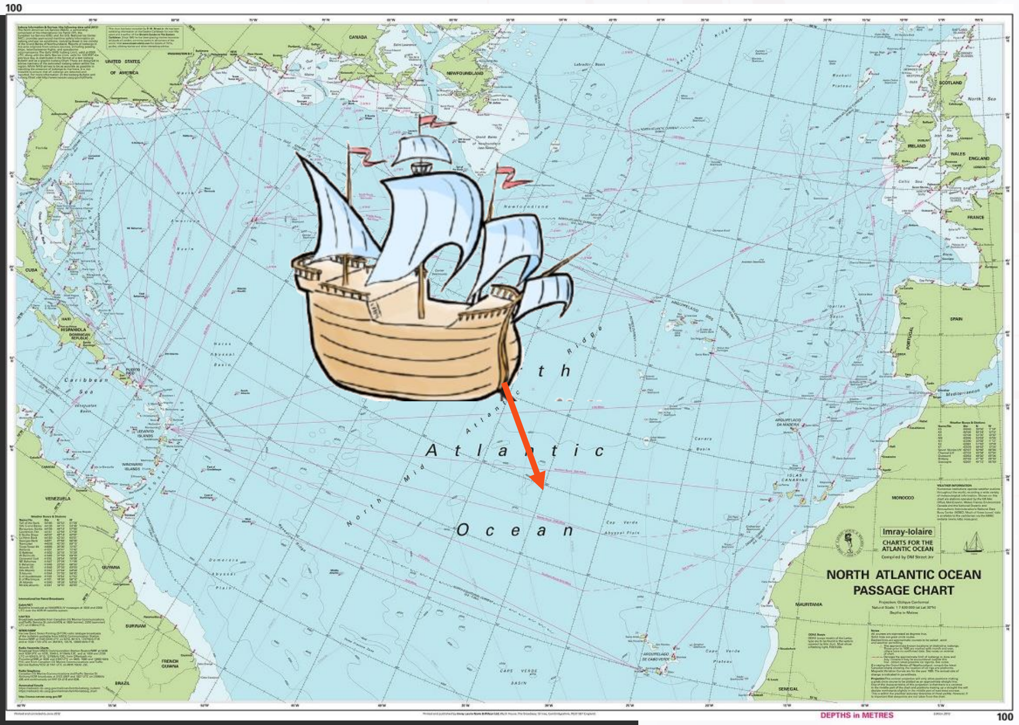
```
#include <stdio.h>

int y = 1;

int main()
{
    int x = 4;
    printf("x is stored at %p\n", &x);
    printf("y is stored at %p\n", &y);
    return(0);
}
```

```
x is stored at 000000000061FE1C
y is stored at 0000000000403010
```



## How a function handles arguments

```c
#include <stdio.h>

void go_south_east(int lat, int lon)
{
    lat = lat - 1;
    lon = lon + 1;
}

int main()
{
    int latitude = 32;
    int longitude = -64;
    go_south_east(latitude, longitude);
    printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
    return 0;
}
```



```
Avast! Now at: [32, -64]
```

*C makes copies of the arguments, before passing them to a function*

## Using pointers

- Obtaining the pointer (address)

```
int x = 4;
printf("x is stored at %p\n", &x);
```

- Declaration

```
int *pointer_to_x;
```

- Assignment

```
pointer_to_x = &x;
```

- Read the contents

```
int value_stored = *pointer_to_x;
```

- Change the contents

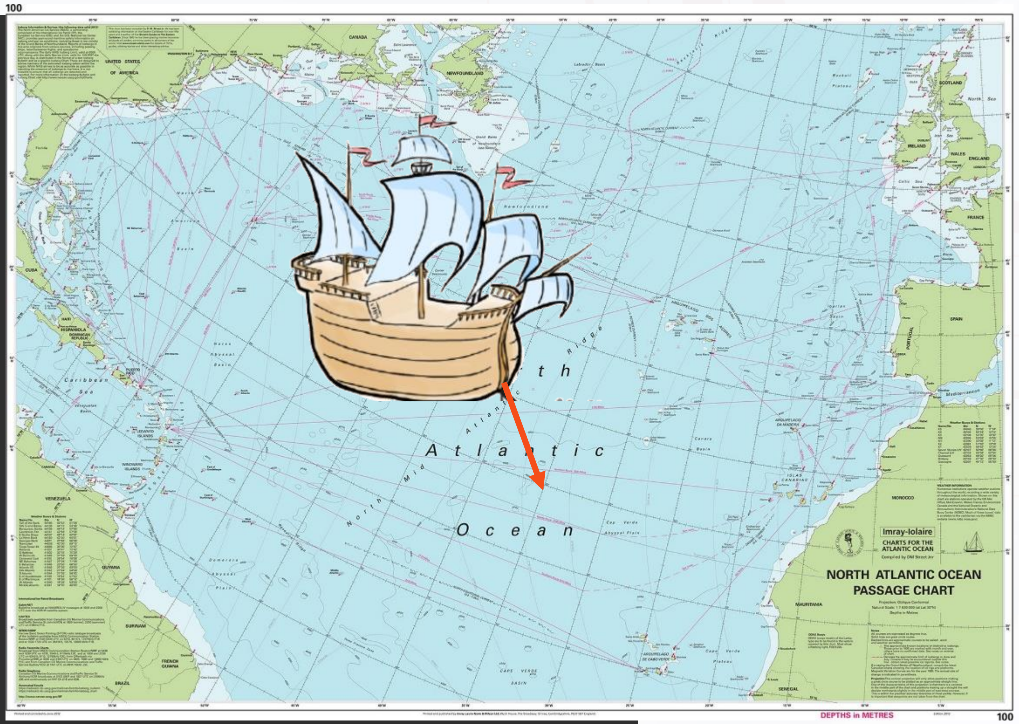
```
*pointer_to_x = 100;
```

*** operator**:
To access the memory addressed by a pointer.

# Passing pointers to a function

```c
#include <stdio.h>

void go_south_east( int *lat , int *lon )
{
    *lat = *lat - 1;
    *lon = *lon + 1;
}

int main()
{
    int latitude = 32;
    int longitude = -64;
    go_south_east( &latitude , &longitude);
    printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
    return 0;
}
```

?
```c
*lat--;
*lon++;
```



```
Avast! Now at: [31, -63]
```

## C Operator Precedence

| Precedence | Operator | Description |
|---|---|---|
| 1 | ++ --<br>()<br>[] | Suffix/postfix increment and decrement<br>Function call<br>Array subscripting |
| 2 | !<br>*<br>&<br>sizeof | Logical NOT<br>Indirection (dereference)<br>Address of …<br>Size of … |
| 3 | * / % | Multiplication, division, and remainder |
| 4 | + - | Addition and subtraction |
| 5 | < <=<br>> >= | Relational operators < and ≤ respectively<br>Relational operators > and ≥ respectively |
| 6 | == != | Relational = and ≠ respectively |
| 7 | && | Logical AND |
| 8 | \|\| | Logical OR |
| 9 | =<br>+= -= | Simple assignment<br>Assignment by sum and difference |

## Actually, we have already used such a function

```c
#include <stdio.h>

int main()
{
    int decks;
    puts("Enter a number of decks");
    scanf("%i", &decks);
    if (decks < 1)
    {
        puts("That is not a valid number of decks");
        return 1;
    }
    printf("There are %i cards\n", (decks * 52));
    return 0;
}
```

*Why does scanf need a pointer argument?*

## How about strings?

```
    char card_name[3];
    puts("Enter the card_name: ");
    scanf("%2s", card_name);
```

Why don't we add the &?

```
void fortune_cookie(char msg[])
{
    printf("Message reads: %s\n", msg);
}

char quote[] = "Cookies make you fat";
fortune_cookie(quote);
```

A string is passed in as a char array

## Array variables

```
#include <stdio.h>

void fortune_cookie(char msg[])
{
    printf("Message reads: %s\n", msg);
    printf("msg occupies %i bytes\n", sizeof(msg));
}

int main()
{
    char quote[] = "Cookies make you fat";
    fortune_cookie(quote);
    return(0);
}
```

sizeof() - a standard C operator (*not a function*) to find how many bytes of space something takes in memory.

```
Message reads: Cookies make you fat
msg occupies 8 bytes
```

Really? Just 8 bytes?