Defining constants in C

```
#define SIZE 10
```

↑ all commands to the preprocessor start with "#".

- preprocessor: initial phase that does textual replacement in the program.
  Here, replaces "SIZE" with 10.
  - the compiler doesn't see "SIZE", it just sees "10".

```
int a[SIZE];
for (int i=0; i<SIZE; i++)
    a[i] = i*5;
```

- Always try to use constants rather than numbers (other than 0) in your program.

Back to structs & pointers

```
struct person {
    char name[100];
    int age;
    int salary
};

struct person *p = malloc(sizeof(struct
                                person));
strcpy(p->name, "John Smith");
p->age = 50;
p->salary = 100000;
```

Rather than always writing "struct person", we can give it a single type name using typedef.

- a way of giving a new name to another type.

```
typedef <existingtype> <newname>;
```

Examples:

```
typedef int SERIAL_NUMBER;
```

- "SERIAL_NUMBER" is another name for int.

optional here.

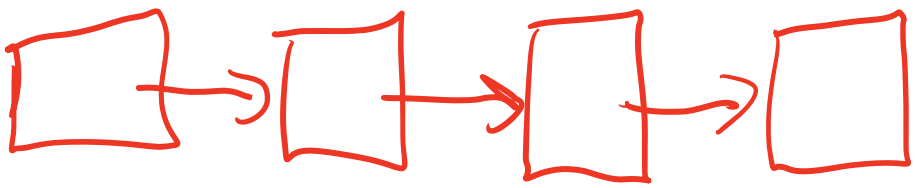
```
typedef struct vehicle {
    int speed;
    char * brand;
} VEHICLE;
```

- introduces the "struct vehicle" type (w/ two fields) and gives the name VEHICLE to that type.

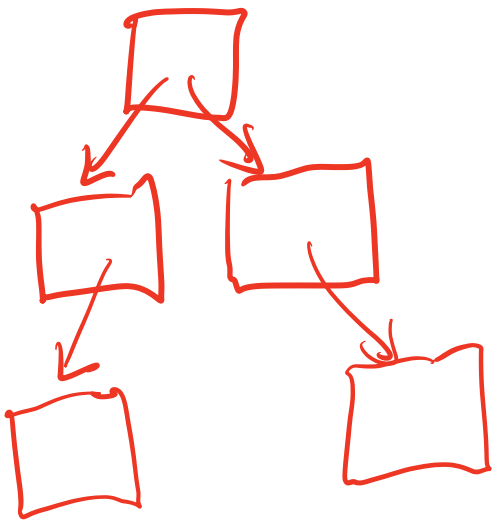
```
VEHICLE V;
V.speed = 20;
V.brand = malloc(50);
                // allocate 50 bytes for
                // the string, including
                // the \0 terminator.

strcpy(V.brand, "Chrysler");
```

Structs may contain pointers to other structs of the same type.

e.g. linked lists:



trees

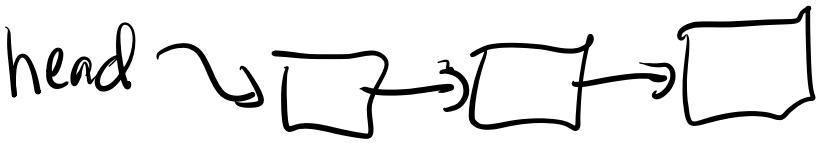


Implementing linked lists:

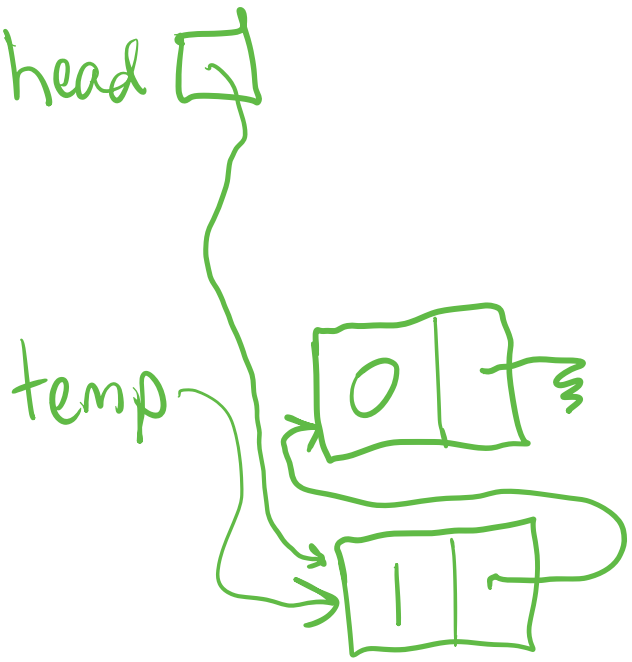
```
typedef struct cell {
    int value;
    struct cell *next;
} ACELL;
```

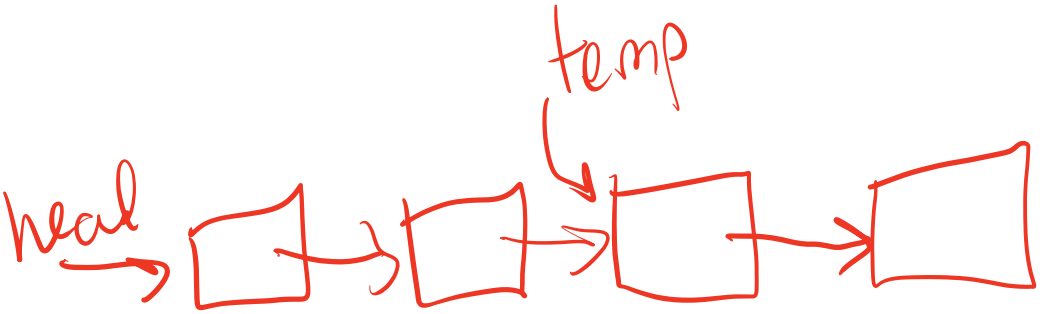
need to write it this way, since the compiler hasn't seen ACELL yet.

Simple code for declaring the head of a list and adding elements to the list.



```
ACELL *head = NULL;
for(int i=0; i<SIZE; i++) {
    ACELL *temp = malloc(sizeof(ACELL));
    temp->value = i;
    temp->next = NULL; //unnecessary
    temp->next = head;
    head = temp;
}
```






```
temp->next->value = i
```

or

```
temp = temp->next
temp->value = i;
```

It is inefficient to search for the end of the list every time you want to add a new element.

- use separate head and tail pointers.

tail->next = . . .

new element

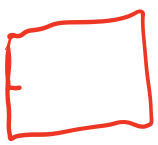
tail = tail->next;

Where in memory does malloc allocate space?

ACELL *temp = malloc(sizeof(ACELL))

What about declaring a variable of a struct type?

ACELL a;

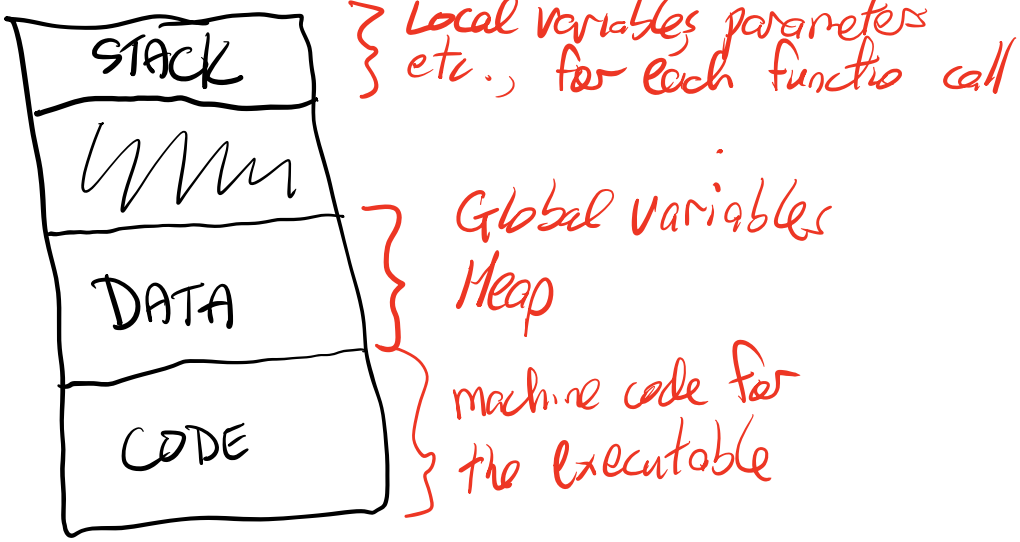


ANSWERS:

malloc allocates space in the heap.

Declaring a variable allocates space in the stack.
- unless the variable is a global variable.

Memory is divided into three areas ("sections", "segments")



Local variables only exist on the stack while the function is running.
- when the function returns, the local variables go away.

Global variables exist for the entire program execution.

Space allocated in the heap exists until the program terminates or the programmer specifies it should be reclaimed.