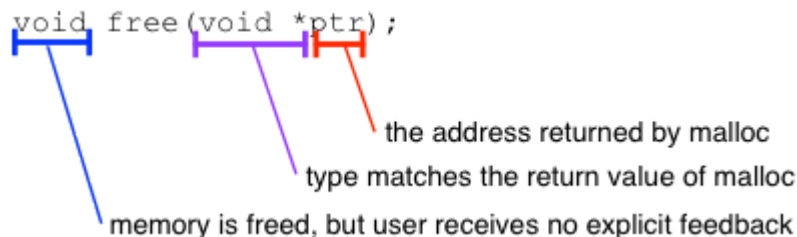


free

free

Unlike memory allocated on the stack (which is freed as soon as the function associated with that stack frame returns), memory on the heap must be explicitly freed by the programmer. For this, the C Standard Library provides the function **free**.

```
void free(void *ptr);
```



The diagram shows the function signature `void free(void *ptr);` with three colored brackets and arrows pointing to annotations:

- A blue bracket under `void` points to the text "memory is freed, but user receives no explicit feedback".
- A purple bracket under `void *` points to the text "type matches the return value of malloc".
- A red bracket under `ptr` points to the text "the address returned by malloc".

The **free** function, whose signature is shown in the figure above takes one argument: the starting address of the memory that needs to be freed. This starting address should match the address that was returned by **malloc**. As a good rule of thumb, every block of memory allocated by **malloc** should be freed by a corresponding call to **free** somewhere later in the execution of the program.

When you free memory, the block of memory that you freed is deallocated, and you may no longer use it. Any attempt to dereference pointers that point into that block of memory is an error—those pointers are dangling. Of course, as with all dangling pointers, exactly what will happen is undefined. When you execute code by hand, and you need to free memory, you should erase the block of memory that is being freed. Specifically, if the code says **free(p)**, you should follow the arrow for **p** (which should reach a block of memory in the heap, or be NULL), and then erase that entire block of memory. If **p** is NULL, nothing happens. If **p** points at something other than the start of a block in the heap, it is an error and bad things will happen (your program will likely crash, but maybe something worse happens).

Note that **free** only frees the memory block you ask it to. If that memory block contains other pointers to other blocks in the heap, and you are done with that memory too, you should free those memory blocks before you free the memory block containing those pointers. In our polygon example, we would free a polygon like this (although better would be to write a **freePolygon** function with the two calls to **free** in it)

3

4

5

6

```
polygon_t * p = makeRectangle(c1, c2);
```

```
//stuff that uses p
```

```
//...
```

```
//done with p and its points
```

```
free(p->points);
```

```
free(p);
```



Note that doing **free(p)** followed by **free(p->points)** would be wrong: **p** would be dangling at the second call, and thus we should not dereference it.



Completed
