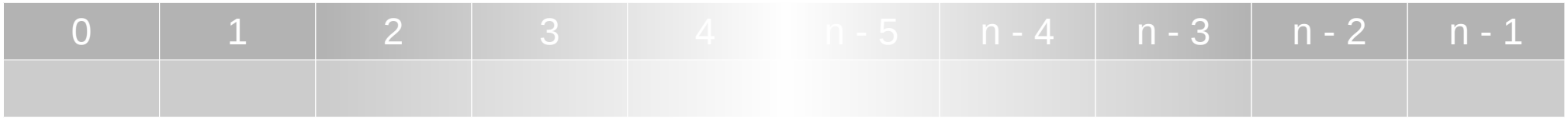
Pointers

(and *malloc()* and *free()* and friends)

How memory works

Memory is divided into ***spaces*** or ***bytes***.



Then how much memory do I have?

4 Gb of RAM = ~4 billion spaces

How we store data into memory

- We can store a lot of things in the memory, but most of these things are too big to fit into one tiny ***space***.
- This is why we created ***data types*** or ***variable types***:
  - `int` can store integers and takes up four ***spaces***;
  - `float` can store floating point numbers and also takes up four ***spaces***;
  - `double` can store floating point numbers that are larger and with more accuracy and takes up 8 ***spaces***;
  - `char` can store a character and takes up only one ***space***;
  - Theoretically, there is an *infinite* number of possible ***types***, including some you may invent yourselves.
- Okay, but what is the address of an `int`? Should we use the address of the first, second, third or fourth ***space***?
  - Always the address of the first ***space***, no matter how many ***spaces*** the ***type*** uses.

How memory is divided

In your programs, the memory is divided into multiple ***zones***:

- The ***stack***;
- The ***heap*** or ***free store***;
- The global/static zone;
- The constant data.

←Not relevant for this presentation

These ***zones*** all work in different ways and have different features.

It's in these ***zones*** that we store variables.

The ***stack*** vs. the ***free store***

The ***stack***

- Is small; does not provide a lot of ***spaces***.
- Lifetime of variables is limited.
- Variables can't be moved or resized.
- Variables have to be declared in advance.
- Really easy to use.

The ***free store***

- Is huge, *really huge*.
- Lifetime of variables is unlimited.
- Variables can me moved and resized.
- Variables do not have to be declared in advance.
- A bit harder to use; requires `malloc` and `free`.

Since the ***stack*** is really easy to use, we stick with it most of the times. However, in some cases, we need to use the ***free store***.

The ***free store*** is extremely powerful.

Tools required to use the ***free store***

Pointers

`malloc`

`free`

`realloc`
(not in GNG1106 but used to move/resize variables)

About pointers

A pointer is a ***type*** of variable, just like an `int`, a `float` or a `char`.

But, instead of containing an integer, a floating point number or a character, its purpose is to contain the address of something on the ***free store***.

Pointers alongside other ***types***

Code	Variable name	Variable ***type***
`int foo;`	foo	int integer
`float bar;`	bar	float floating point number
`char kong;`	kong	char character
`int* tok;`	tok	int* pointer to an `int`
`float* ding;`	ding	float* pointer to a `float`
`char* sit;`	sit	char* pointer to a `char`

Pointers are variables, just like other ***types***. They work the same way and are assignable the same way.

For every ***type*** of variable, there exists one ***type*** of pointer. Even for the ones you create yourselves.

More on pointers and the ***free store***

- Like all other basic variables ***types*** like `int` or `float` or `char`, we usually put the pointers on the ***stack***, even though they contain the addresses of variables on the ***free store***.
- We cannot access variables that are on the ***free store*** directly; we must always go through a pointer that is on the ***stack***.

Good tip to avoid being mixed up:

- How other people usually declare pointers:

```
int *ptr;
```

- How I declare pointers:

```
int* ptr;
```

Notice where the space is. Both notations are valid but I found the second one to be more intuitive. This is how I understood pointers.

Using pointers

With pointers, *just like for any other **type** of variable*:

- Once you declare a pointer:
  ```
  int* ptr;
  ```
- You may declare another pointer with the same value:
  ```
  int* anotherptr = ptr;
  ```
- You may operate on the pointer:
  ```
  ptr++;
  ptr = 2 * ptr;
  ```
  ***Warning:** be careful when operating with pointers.*

Using the values pointed by pointers

Pointers are cool but useless if we don't know how to use the values to which our pointers... point. aka, how do I use the variables on the ***free store*** and not just the pointers?

— *Easy! Use the... star.*

Yes, here again we use a star, but this star has a *completely different use*. It's used to represent the variable that is on the ***free store***.

- Divide by 2 the pointer:

```
ptr = ptr / 2;
```

- Divide by 2 the value pointed by the pointer:

```
*ptr = *ptr / 2;
```

Not mixing * and *

- Declare a pointer:

```
int* ptr;
```

- Use the value pointed by a pointer:

```
*ptr;
```

That's it.

Thou shalt not use stars anywhere else!

Using the ***free store***

- To put a variable on the ***free store***, you must allocate ***spaces*** for it using `malloc`. Let's say you want to put an `int` on the ***free store***. Well you know that an `int` uses 4 spaces. So you write:

```
malloc(4);
```

- What if you don't know the size of the ***type*** of the variable you want to put on the ***free store***? You just use `sizeof()`:

```
malloc( sizeof(int) );
```

- Now you must put the address of your newly-allocated block of memory in a pointer or else you won't be able to use it. This is the whole thing:

```
int* ptr = malloc( sizeof(int) );
```

- Now you can manipulate your variable on the ***free store*** using the pointer!
- Once you're done, just make sure you `free` the memory on the ***free store*** so it can be used again:

```
free(ptr);
```

Hijacking pointers

- We know that pointers are variables. Pointers are variables that contain addresses. We usually put the address of a variable on the ***free store*** in a pointer.
- But pointers may also point to variables that are not on the ***free store***.
  In fact:
  - A pointer can point to a variable on the ***stack***;
  - A pointer can be put on the ***free store*** just like any other variable;
  - A pointer on the ***free store*** can point to a pointer on the ***stack*** (very rare).
- To point to a variable on the ***stack***, you declare the pointer as usual, then you assign it the address of the variable using the operator `&`:

  ```
  int* ptr = &foo;
  ```

  Now `ptr` contains the address of `foo` which is a variable on the ***stack***.

To summarize...

- Pointers are variables, too.
- The pupose of a pointer is to hold an address the same way the purpose of an `int` is to hold an integer.
- Pointers become necessary when you want to use the free store.
- Pointers can point everywhere, not necessarily on the free store.