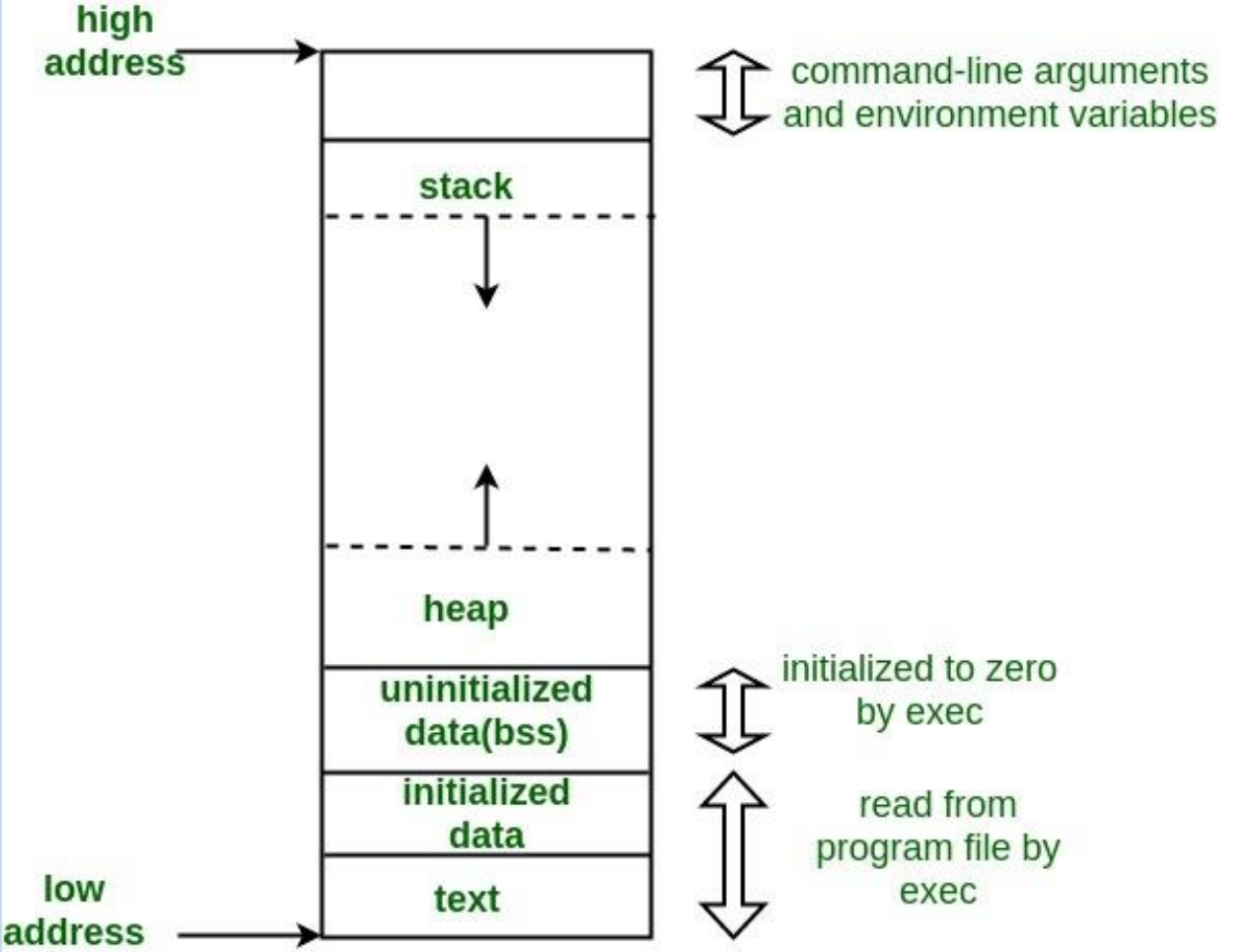
Unit 1

Part 1

Pointers, Dynamic memory allocation

Memory Layout of C Programs



Source: https://www.geeksforgeeks.org/memory-layout-of-c-program/

Pointers

- Every variable in C has a name and a value associated with it. When a variable is declared, a specific block of memory within the computer is allocated to hold the value of that variable. The size of the allocated block depends on the type of the data.

```
int x = 10;
```

- When this statement executes, the compiler sets aside 2 bytes of memory to hold the value 10. It also sets up a symbol table in which it adds the symbol *x* and the relative address in memory where those 2 bytes are set aside.

Pointers

- Thus, every variable in C has a value and an also a memory location (commonly known as address) associated with it. Some texts use the term **rvalue** and **lvalue** for the value and the address of the variable respectively.
- The rvalue appears on the right side of the assignment statement and cannot be used on the left side of the assignment statement.
- Therefore, writing *10 = k;* is illegal.

Declaring Pointer Variables

- A pointer is a variable that contains the memory location of another variable.
- The general syntax of declaring pointer variable is

```
data_type  *ptr_name;
```

- The '*' informs the compiler that *ptr_name* is a pointer variable and *data_type* specifies that it will store the address of *data_type* variable.
- The & operator retrieves the lvalue (address) of *x*, and copies that to the contents of the pointer ptr.

Dereferencing a Pointer Variable

- We can "dereference" a pointer, i.e. refer to the value of the variable to which it points by using unary '*' operator as in *ptr.
- That is, **ptr = 10*, since 10 is value of *x*.

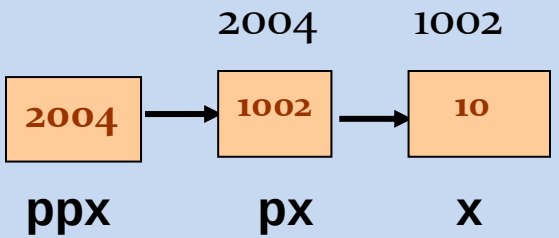
Pointer to Pointers

- You can use pointers that point to pointers. The pointers in turn point to data (or even to other pointers). To declare pointers to pointers just add an asterisk (*) for each level of reference.
- For example, if we have:

```
int x=10;
int *px, **ppx;
px = &x;
ppx = &px;
```

Now if we write,

```
printf("\n %d", **ppx);
```



This would print 10, the value of `x`.

Pointer Expressions and Arithmetic

- Pointer variables can also be used in expressions. For example,

```
int num1=2, num2= 3, sum=0, mul=0, div=1;
int *ptr1, *ptr2;
ptr1 = &num1, ptr2 = &num2;
sum = *ptr1 + *ptr2;
mul = sum * *ptr1;
```

- We can add integers to or subtract integers from pointers as well as subtract one pointer from the other.
- We can compare pointers by using relational operators in the expressions. For example p1 > p2 , p1==p2 and p1!=p2 are all valid in C.

Pointer Expressions and Arithmetic

- When using pointers, unary increment (++) and decrement (--) operators have greater precedence than the dereference operator (*).
- Therefore, the expression *ptr++ is equivalent to *(ptr++).
- The expression will increase the value of ptr so that it now points to the next element.
- In order to increment the value of the variable whose address is stored in ptr, write (*ptr)++.

Null Pointers

- A ***null pointer*** is a special pointer value that is known not to point anywhere. This means that a NULL pointer does not point to any valid memory address.
- To declare a null pointer you may use the predefined constant NULL.

```
int *ptr = NULL;
```

- It is used in situations if one of the pointers in the program points somewhere some of the time but not all of the time.
- In such situations it is always better to set it to a null pointer when it doesn't point anywhere valid, and to test to see if it's a null pointer before using it.

Generic Pointers

- A generic pointer is pointer variable that has void as its data type.
- The generic pointer can be pointed at variables of any data type.
- It is declared by writing

```
void *ptr;
```

- You need to cast a void pointer to another kind of pointer before using it.
- Generic pointers are used when a pointer has to point to data of different types at different times.

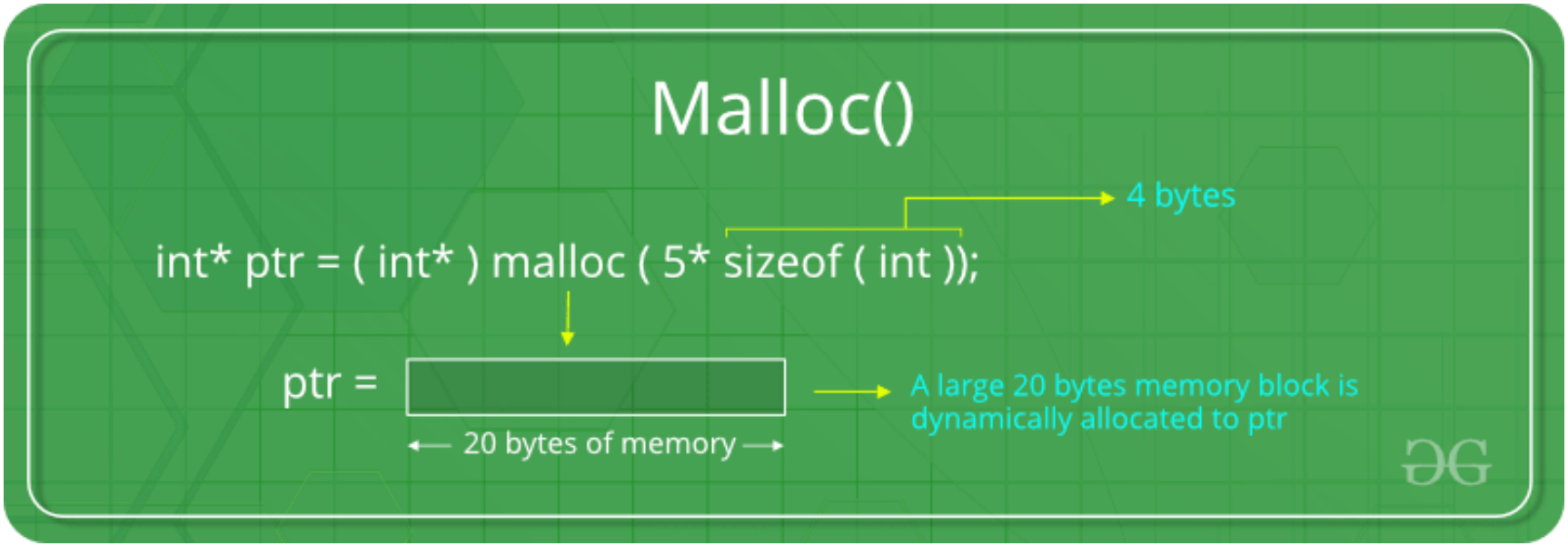
Passing Arguments to Functions using Pointers

- The calling function sends the addresses of the variables and the called function must declare those incoming arguments as pointers.
- In order to modify the variables sent by the caller, the called function must dereference the pointers that were passed to it.
- Thus, passing pointers to a function avoids the overhead of copying data from one function to another.

Dynamic memory allocation

- Allocating memory during runtime/execution of the program.
- Allows to allocate only required amount of memory.
- Allocated from heap segment of the program's memory.
- Dynamically allocated memory need to be freed(de-allocated) explicitly after use.
- Library functions are available for allocation and deallocation.

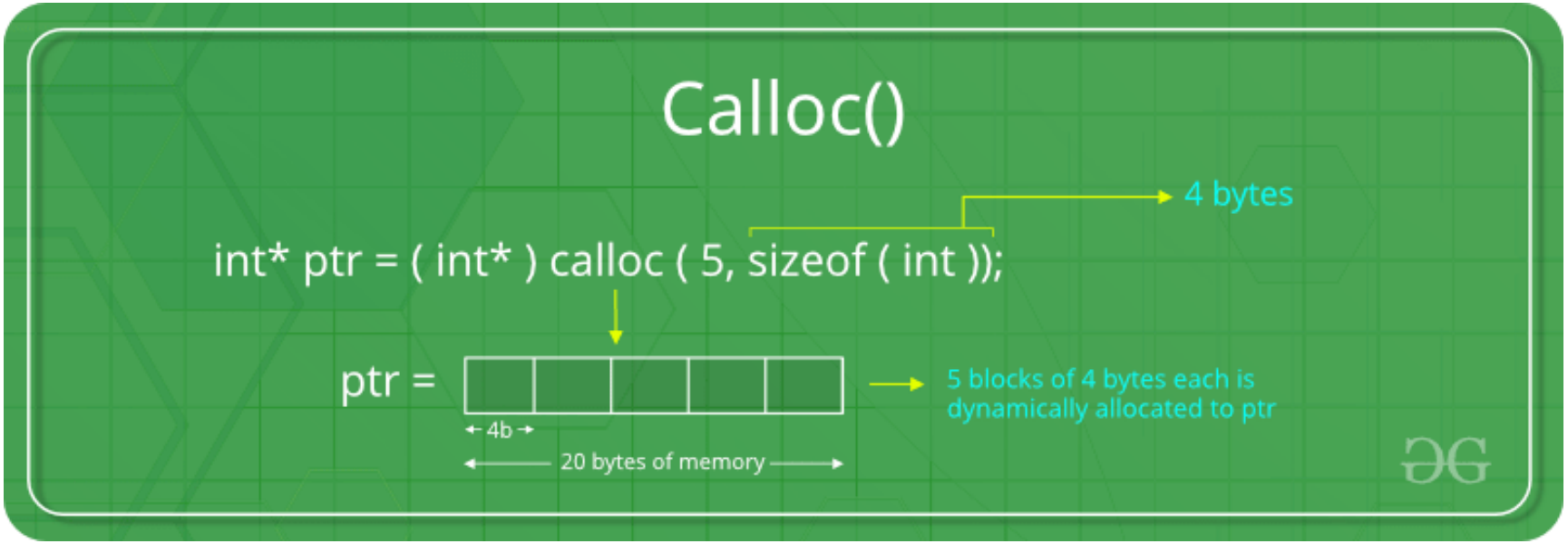
malloc



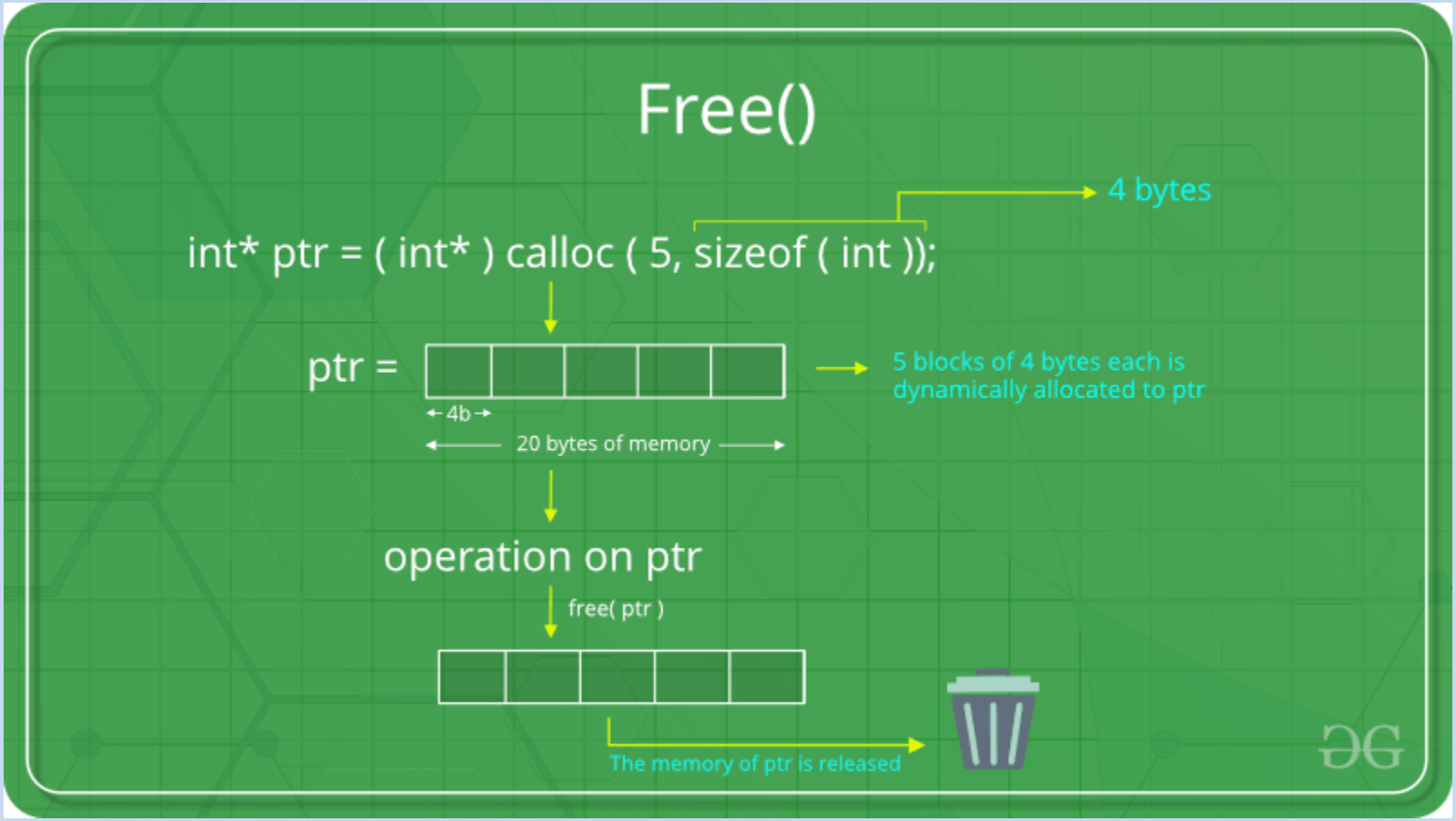
Source: https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

calloc()

- **Source:** https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/



Freeing memory



Reallocating memory