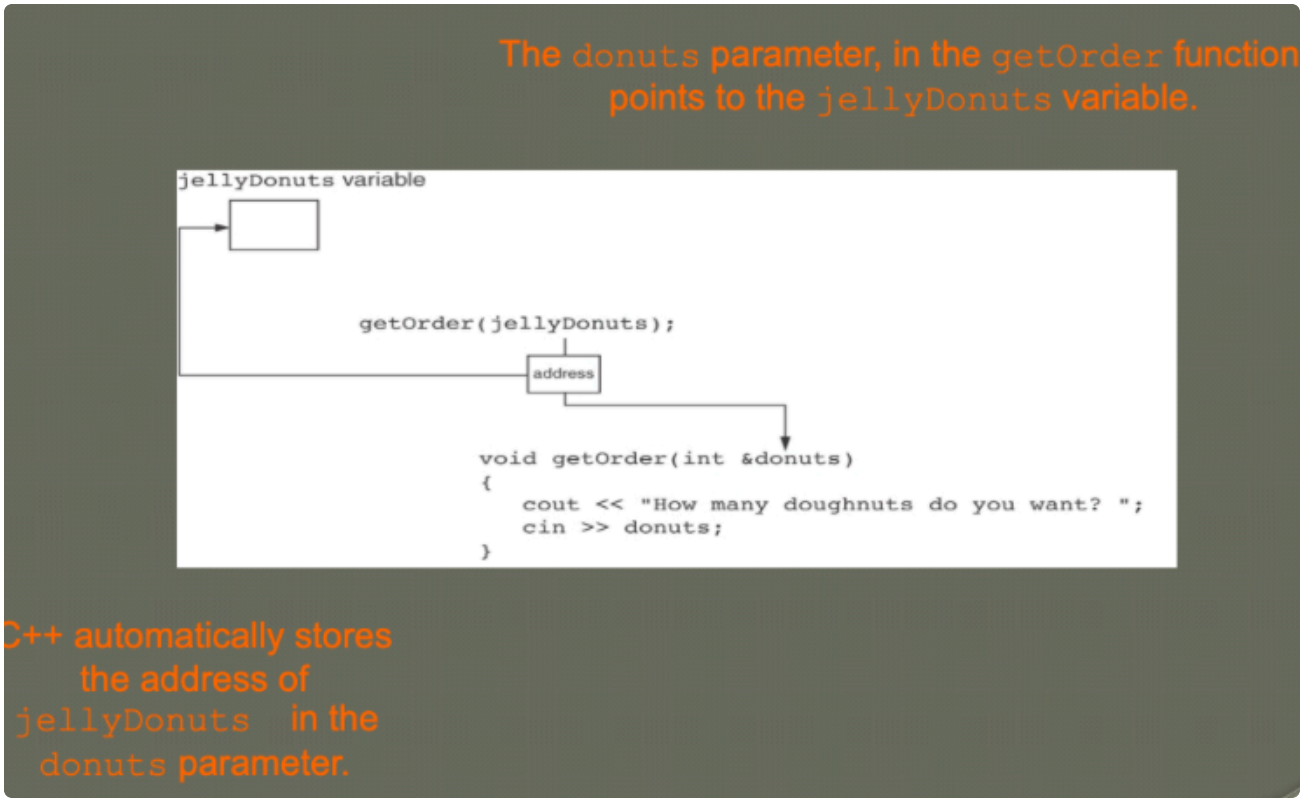
Intro to Pointers

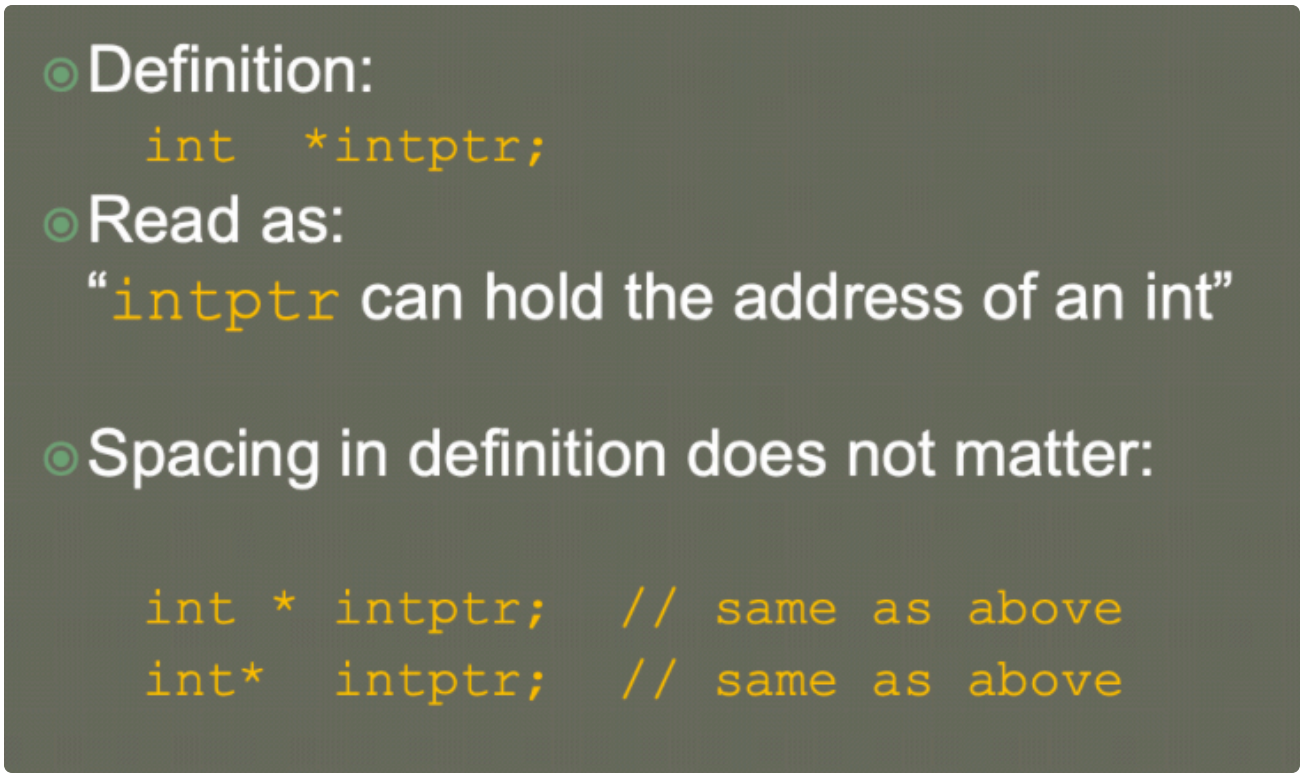
- The address variable `&` returns the address of a variable
  - Each byte has an unique address
  - Each variable is stored at a unique address
  - *Variables are made of bytes...*
- *Tip on finding byte location*:

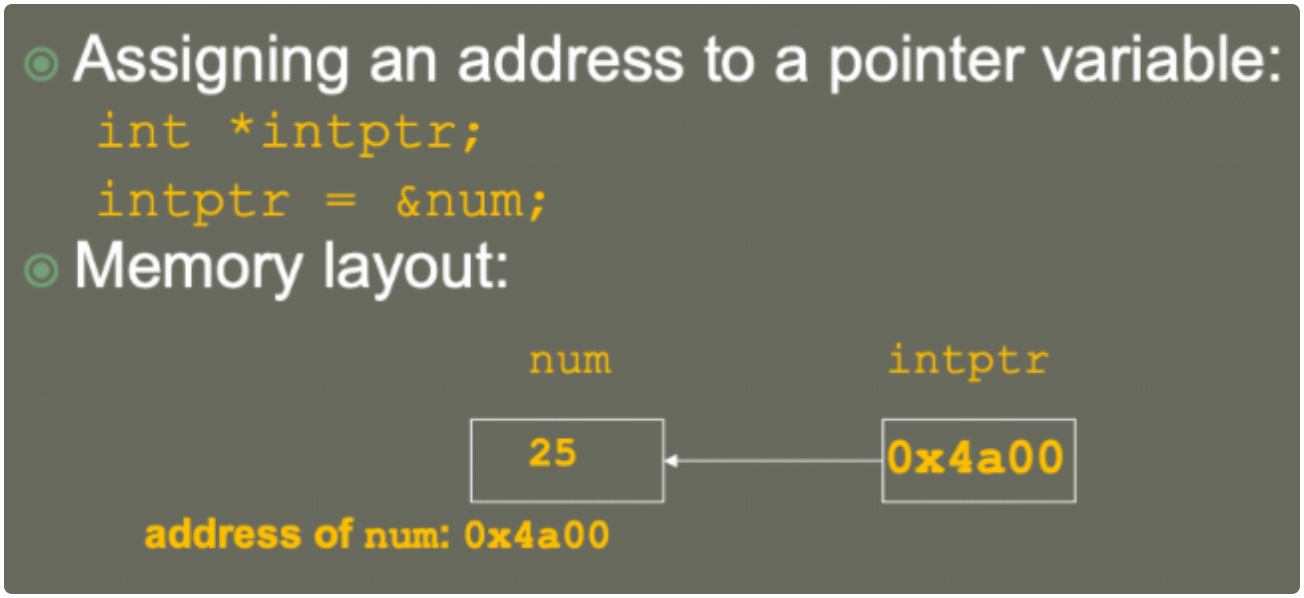
```
int num = -99;
cout << &num; // prints address
              // 0x22FF44
```

- This follows the same idea of reference variables

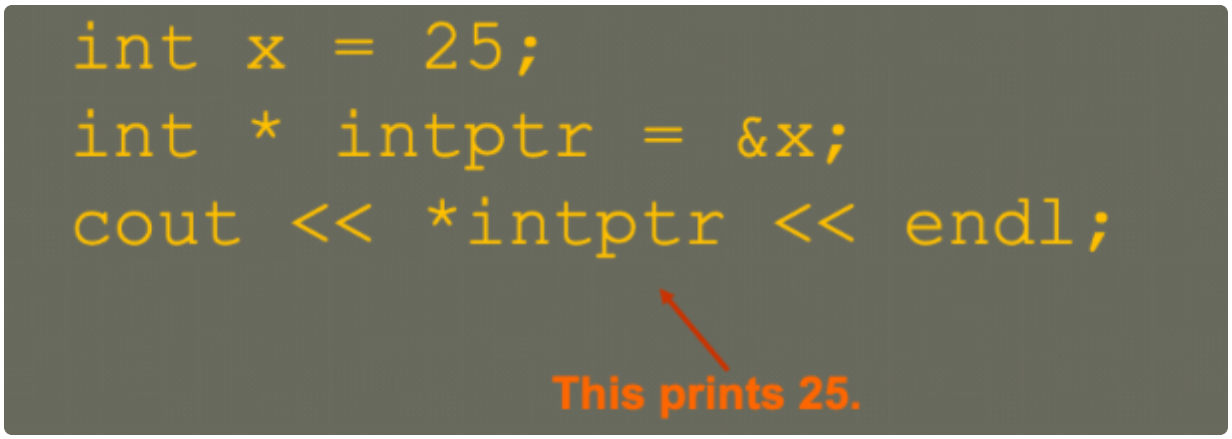


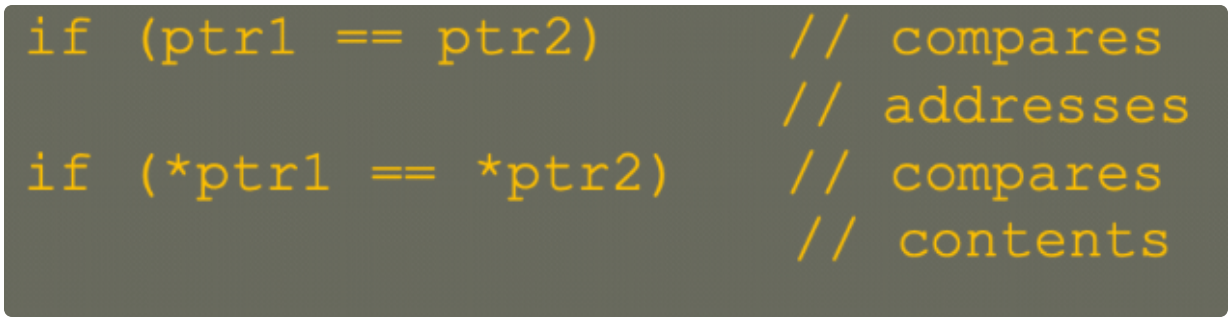
- Pointers are used to **Dynamically Allocate Memory**!
  - When running a program with unknown amount of data, dynamic memory allocation allows you to create/limit variables, arrays and complex data structures in memory while running the program
  - Pointers are useful to manipulate arrays and C style strings
  - In Object Oriented Programming, pointers are used for creating and working with objects and sharing access to these objects

- **Pointer variable**: Often called a **pointer**, is a variable that holds addresses
  - They are variables that points to another piece of data
  - Considered 'low-level' as compared to arrays and reference variables
  - You must find the address you want to store and correctly use it
- 

- 

- `(*)` operator dereferences a pointer
  - If called via `cout` it will allow you to access the item it points to

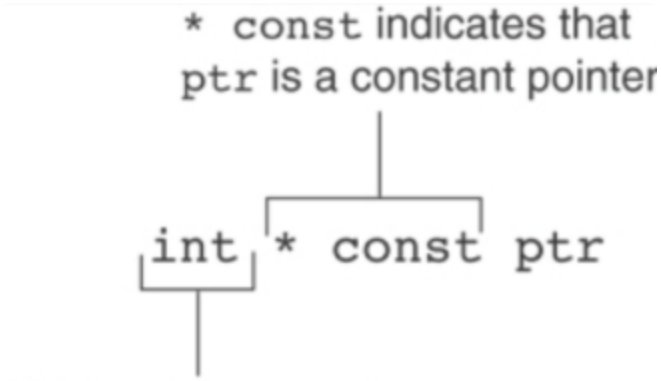
- 

- You can not mix pointer data types, if the pointer is initialized as a int, it must point to an int.
  - You can also test for a valid/existing address using `if(!ptrNameHere)`
- Comparing the data **IN** pointers is not the same as comparing the *CONTENTS* pointed at by pointers
  - pointers returns content
  - regular standalone variables return memory location
  - *kinda funny how they the opposite*
  - 

- Pointers can also be the return type of functions!
  - Should only be used to data that was passed to the function as an argument or dynamically allocate memory.
- **Stack Memory** is allocated automatically on function call and deallocated automatically when function ends
  - Separate section holds `static` data
  - Stack memory also hold global variables
- **Heap Memory** explicitly requests the allocation of a memory "block" of a certain size, and remains allocated until its requested to be deallocated.
- **Dynamic Memory Allocation** can allocate storage for a variable while the program is running
  - Computers can return the addresses of newly allocated memory
  - `new` can be used to allocate memory

- ```
  double *dptr;
  dptr = new double;
  ```
  - The first line essentially creates the pointer, but does not have any usable or valid memory.
    - Stored on the Stack
  - The second line allocates the actual memory for double and makes the pointer point to the double
    - Stored on the heap
- `delete` can be used to free dynamic memory
  - ```
    delete fptr;
    ```
  - ```
    delete [] arrayptr; //will talk about
                        // arrays next time
    ```
  - ONLY USE WITH DYNAMIC MEMORY
- It is good practice to store `NULL` in a pointer after using delete on it as it prevents code from accessing the memory after its used
  - MORE SIMPLE EXPLANATION: Worker (being the data) is fired, so you ask them do the job or they will sabotage
- Pointers also work with classes, similar to how they usually are
- Must use a *dereference* pointer variable if trying to be accessed
  - ```
    cout << (*stuPtr).studentID;
    ```
  - Can also be written like this, which is a lot cleaner
  - ```
    cout << stuPtr->studentID;
    ```
- To store a constant in a pointer, it must be stored in a **pointer-to-const**
    ```

-
```
const int SIZE = 6;
const double payRates[SIZE] =
      { 18.55, 17.45, 12.85,
        14.97, 10.35, 18.89 };
```
-
```
void displayPayRates(const double *rates, int size)
{
   for (int count = 0; count < size; count++)
   {
      cout << "Pay rate for employee " << (count + 1)
           << " is $" << *(rates + count) << endl;
   }
}
```
The parameter *rates*, is a pointer to `const double`.
- A **constant pointer** is a pointer that is initialized with an address that can not be pointed to anything else



- A **constant pointer to a constant** is a pointer that points to a constant and the pointer can not point to anything but what its pointing to.
  - what about a pointer pointing to a constant...
  -

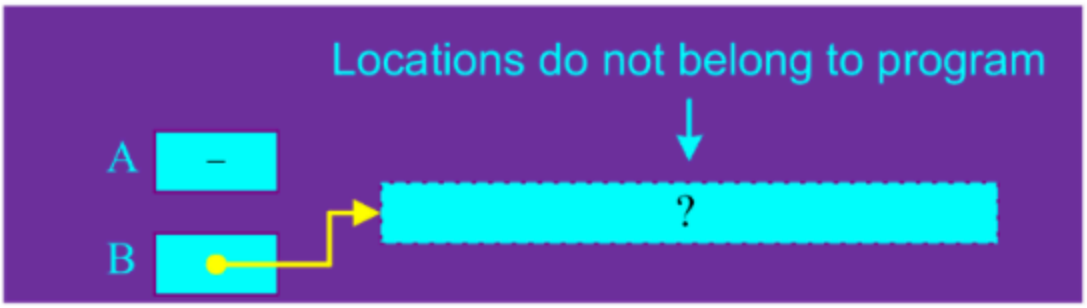
Type	Example Declaration	Can you change the **data** (`*ptr`)?	Can you change the **pointer** (`ptr`)?
Pointer to Constant	`$const int *ptr;`	**No** ❌	**Yes** ✅
Constant Pointer	`$int * const ptr;`	**Yes** ✅	**No** ❌
Constant Pointer to Constant	`$const int * const ptr;`	**No** ❌	**No** ❌

Pointers and Dynamic Memory Allocation

- `&` aka. **address-of operator**
- In C++ there are three types of memory used in programs
  - **Static memory** - where global and static variables live
    - Allocated at compiler time
  - **Heap memory** - Where dynamically memory is allocated during execution
    - Unnamed variables
    - Explicitly allocated during execution by code written by programmer, using `new` and `delete`
  - **Stack memory** is used by automatic (local variables)
    - Auto variables
    - Function parameters
    - Automatically created when functions are called and are destroyed when returning from functions.
- If memory is available, in an area called the heap (or free store) `new` allocates the requested amount of memory and returns a pointer to the memory allocated.
  - No memory available = program termination
  - Exists until `delete` operator destories it
- It is good to store `NULL` in a pointer after using delete on it
  - It prevents code from unintentionally using the pointer after that area is freed
  - Prevents errors from double deletion
  - Improper pointer handling can cause memory leak and other problems
- **Dangling Pointers** are pointers that point de-allocated memory

```
int *A = new int [5]{0,1,2,3,4};
int *B = A;
```


```
delete [] A;
B[0] = 1; // illegal!
```



- A function ONLY RETURN A POINTER IF:
  - Data that was passed to the function was a reference argument
  - To dynamically allocate memory