WELCOME TO
🎉 DATA STRUCTURES & ALGORITHMS
🎉

LESSON OVERVIEW

- Meet your tutor and icebreaker
- Course overview
- Lab content
  - Input and Output
  - Factorial problem
  - Pointers and references
  - Introduction to GTest
  - Classes

TOM GOLDING

- Studying a bachelor of Computing Science with a major in Enterprise Systems Development
- Aspiring software engineer
- Loves to play the guitar

Contact: Thomas.golding@uts.edu.au

COURSE OVERVIEW

- The C++ language
- Pointers and references
- Big O notation
- Complexity theory
- Basic data structures and algorithms for them:
  - Lists
  - Arrays
- Sorting algorithms
- Advanced data structures and algorithms for them:
  - Binary tree data structure
  - Hashing, sets and maps
  - Graph data structure

COURSE OVERVIEW

ASSESSMENTS

- Programming assignment 1 (12/04/2024) 20%
- Programming assignment 2 (24/05/2024) 30%
- Weekly exercises (week 2 – week 12) 20%
- Final exam (Exam period) 30%

LAB TIME!!!

Input and output

`std::cout`

- You can access `cout` with `#include <iostream>`
- `std` is a namespace which stands for “standard library”
- `cout` stands for “character output”
- Similar to Java’s System.out.println(), it allows you to print to the console

Input and output

```
std::cout << ”Hello World”;
```

- The << symbol is the insertion operator
- Sending data from the right side to the left into the character output

Input and output

```
std::cout << ”Hello World\n”;
```

```
std::cout << ”Hello World” << std::endl;
```

- `endl` stands for “end line”
- Both `\n` and `std::endl` do the same thing
- Except, `std::endl` flushes the buffer, whilst `\n` doesn’t

Input and output

`std::cin`

- `std` is a namespace that stands for “standard library”
- `cin` stands for “character input”
- Similar to Java’s Scanner object, it allows you to write to the console

Input and output

```
std::string userName {};
std::cin >> username;
```

- The `>>` symbol is the extraction operator
- Sending our inputted data from `cin` to our variable `username` to be stored

Now time to try out the input/output exercises

Feel free to give me a holler if you need to ask some questions 😎

Factorials

$$n!$$

- A factorial is the multiplication of every number from 1 to n

Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

- A factorial is the multiplication of every number from 1 to n

Factorials

$$0! = 1$$

- A factorial is the multiplication of every number from 1 to n
- Except in the case of $0!$, but why?

Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

- Well we can simplify this problem into

Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$5! = 5 \times 4!$$

- Well we can simplify this problem into
- And further into

Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$5! = 5 \times 4!$$

$$4! = \frac{5!}{5}$$

- Well we can simplify this problem into
- And further into

Factorials

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

$$5! = 5 \times 4!$$

$$4! = \frac{5!}{5}$$

- Well we can simplify this problem into
- And further into
- Therefore, if we had

Factorials

$$1! = 1$$

- Well we can simplify this problem into
- And further into
- Therefore, if we had

Factorials

$$1! = 1$$

$$1! = 1 \times 0!$$

- Well we can simplify this problem into
- And further into
- Therefore, if we had

Factorials

$$1! = 1$$

$$1! = 1 \times 0!$$

$$0! = \frac{1!}{1} = 1$$

- Well we can simplify this problem into
- And further into
- Therefore, if we had

Now time to try out the factorial exercise

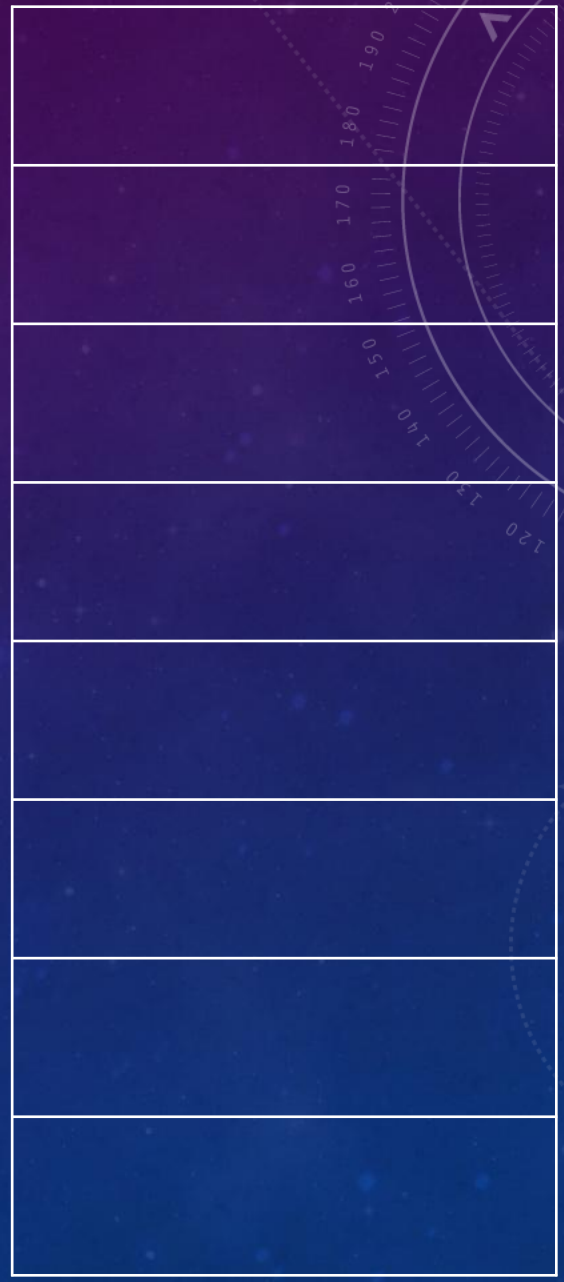
Feel free to give me a holler if you need to ask some questions 😎

Variables

- A variable is a named storage in computer memory that holds a value

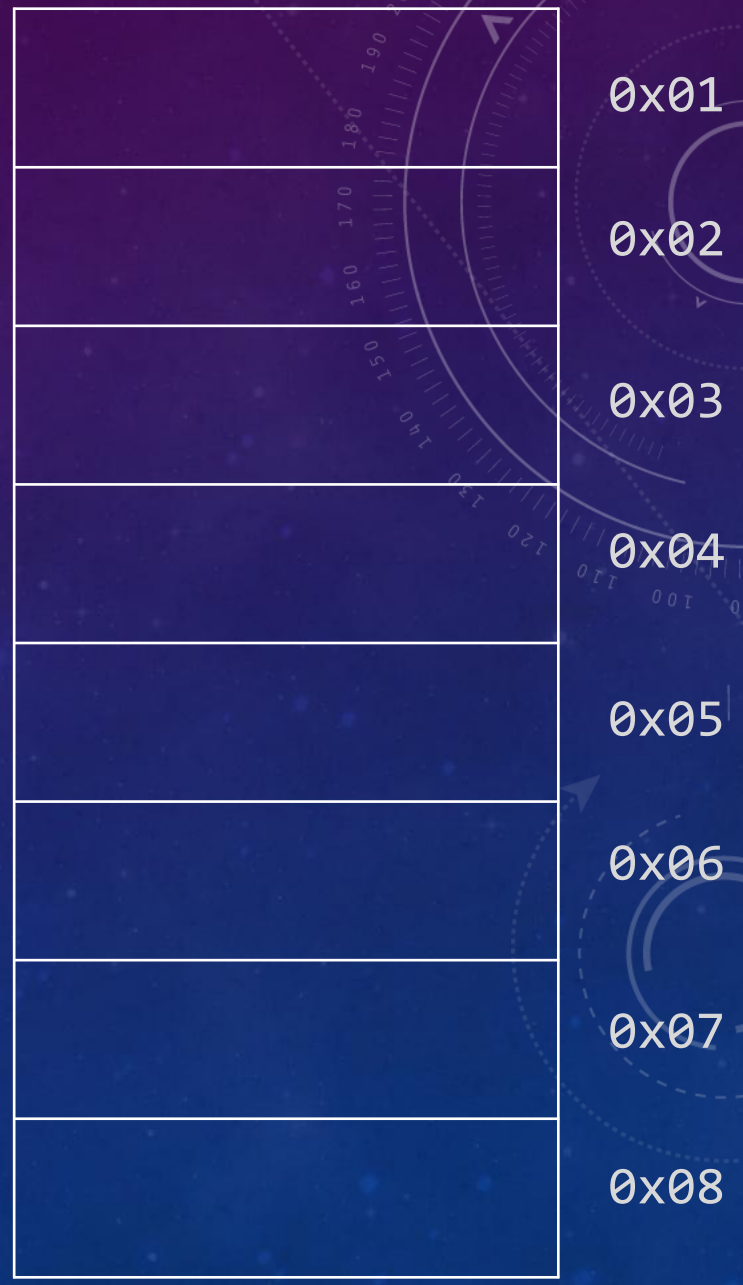
Variables

- A variable is a named storage in computer memory that holds a value
- Variables are stored in a contiguous line of boxes in the RAM



Variables

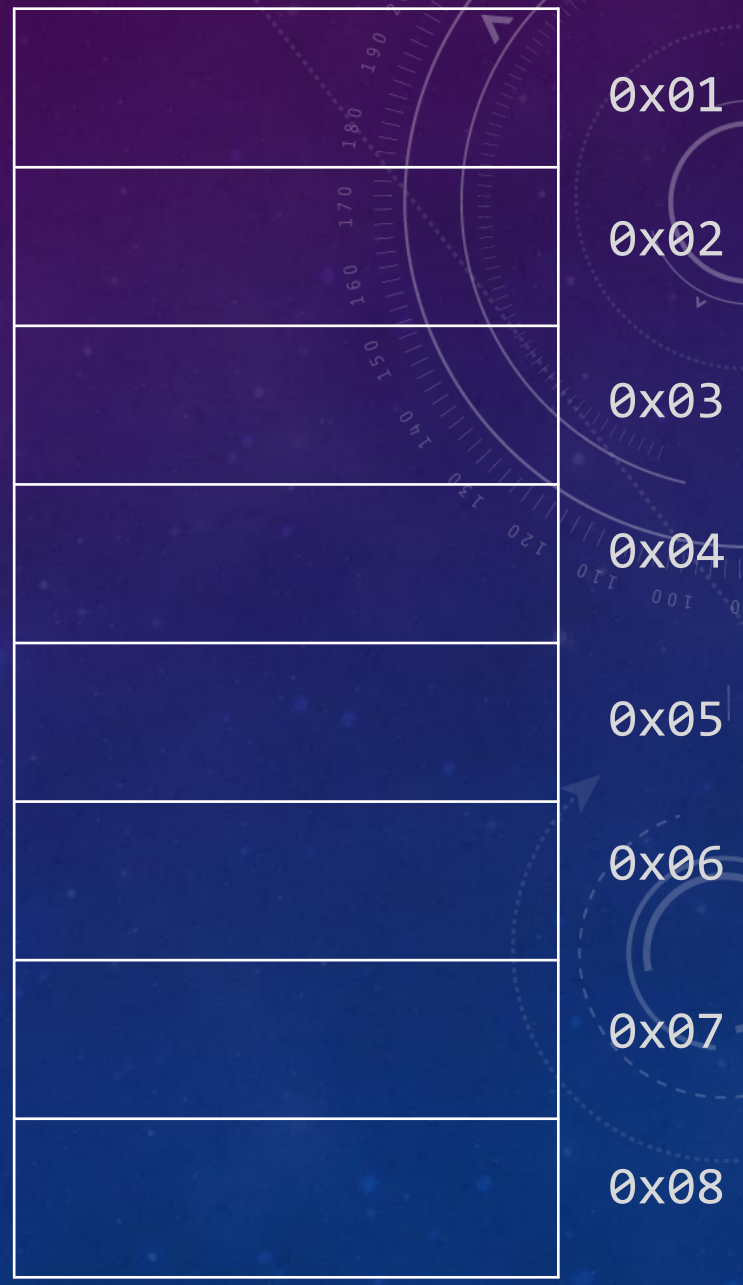
- A variable is a named storage in computer memory that holds a value
- Variables are stored in a contiguous line of boxes in the RAM
- Each box is associated with a unique hexadecimal address



Declaring variables

```
int age;
```

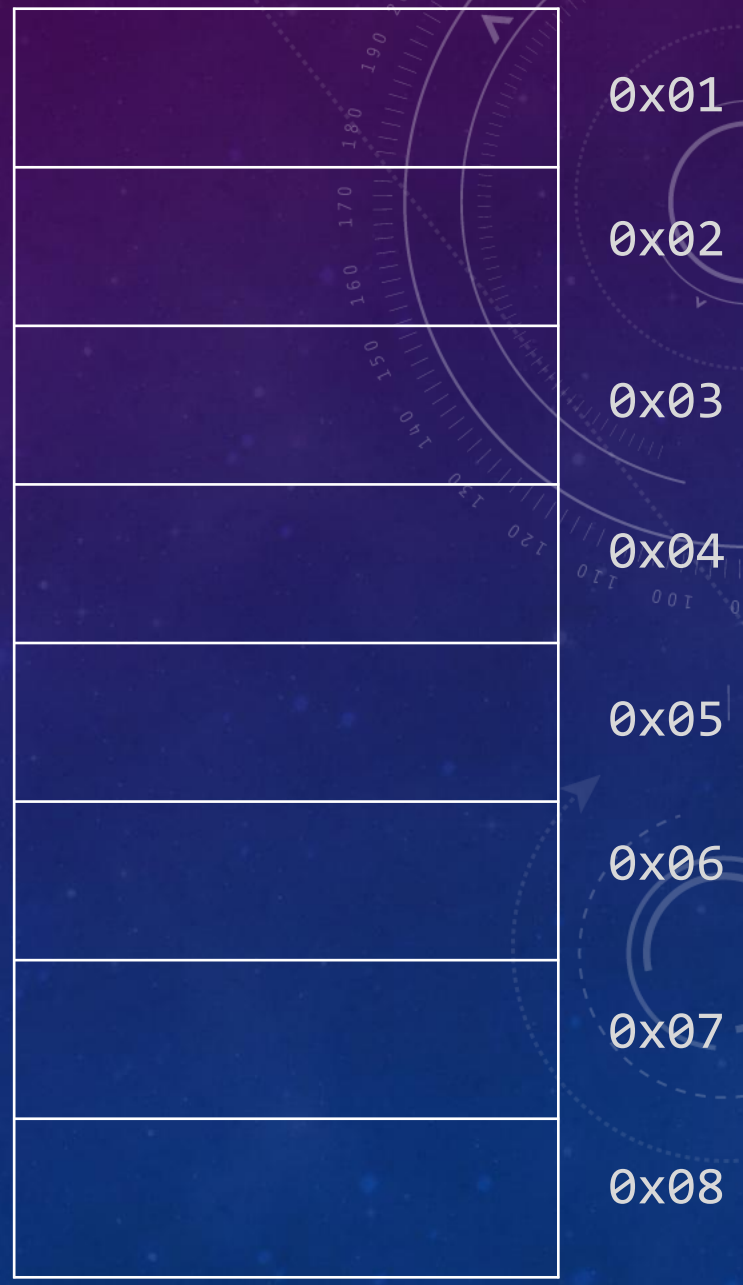
- When you declare a variable, it is randomly assigned a free box



Declaring variables

```
int age;
```

- When you declare a variable, it is randomly assigned a free box. So now `age` has the address `0x02`.

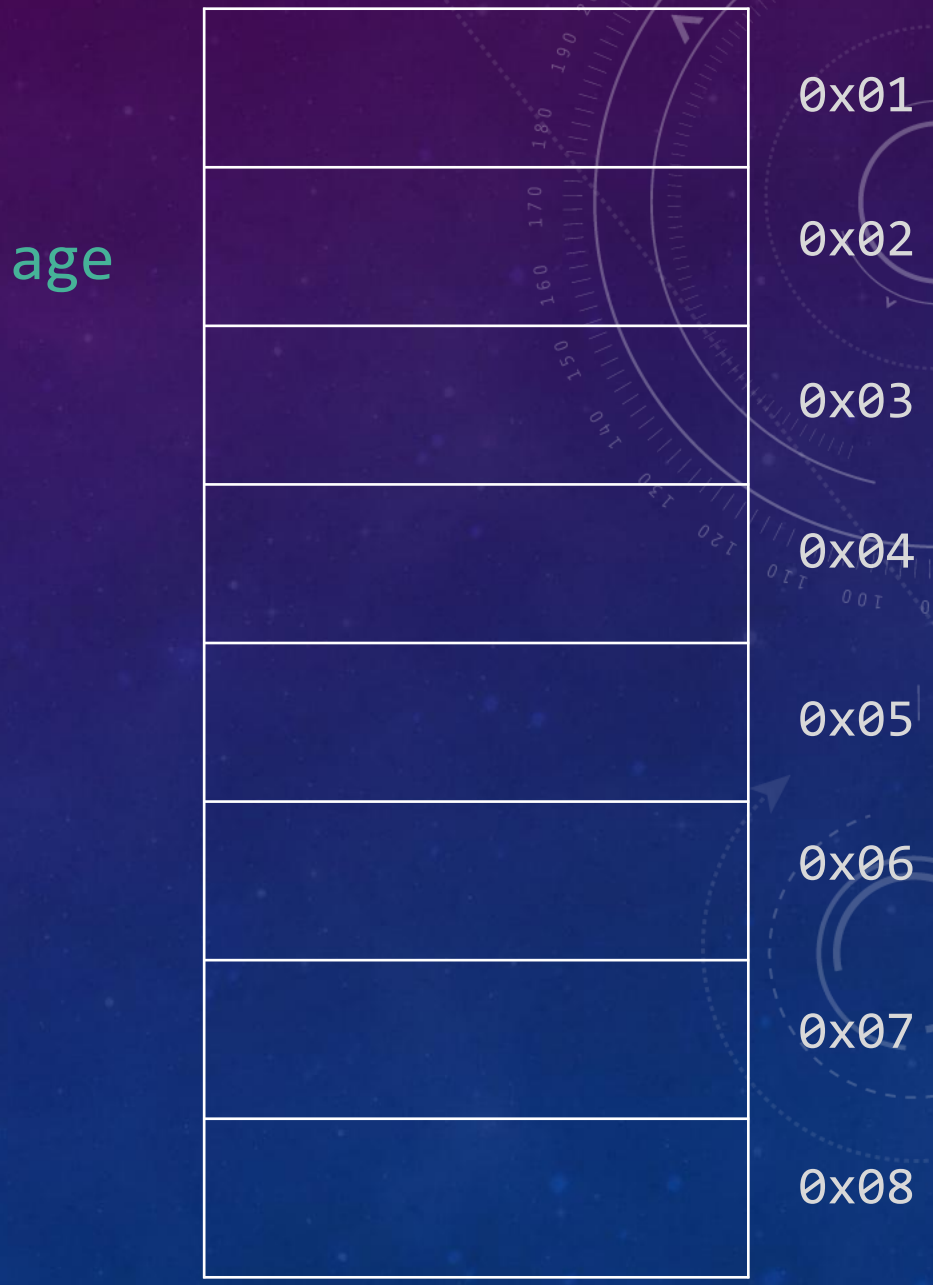


Declaring variables

```
int age;
```

```
int age {24};
```

- When you declare a variable, it is randomly assigned a free box. So now `age` has the address `0x02`.
- When you assign a value to `age`, the memory occupying that space will contain that data



Declaring variables

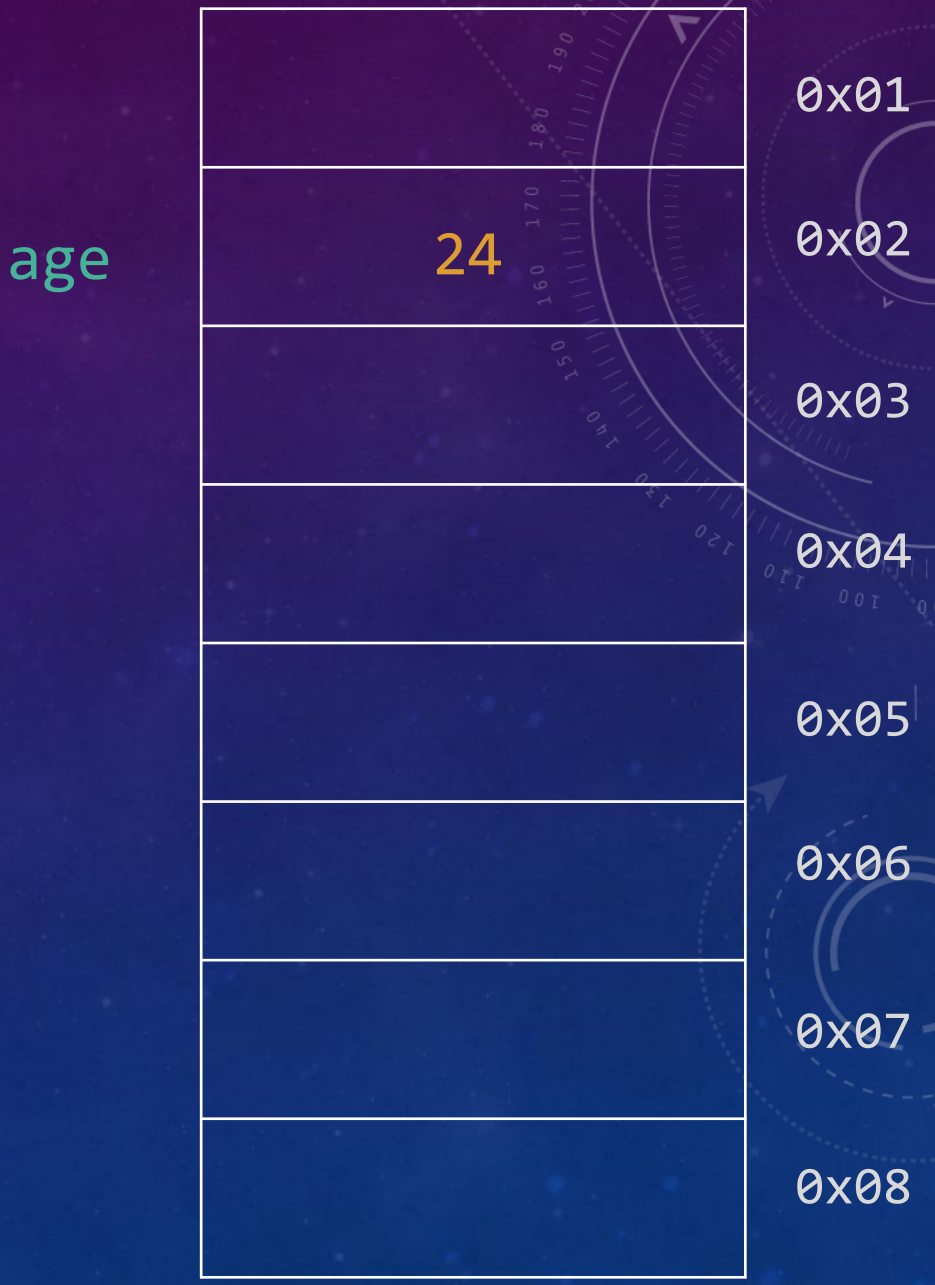
```
int age;

int age {24};
```

- When you declare a variable, it is randomly assigned a free box. So now `age` has the address `0x02`.
- When you assign a value to `age`, the memory occupying that space will contain that data

References

```
int age {24};
```

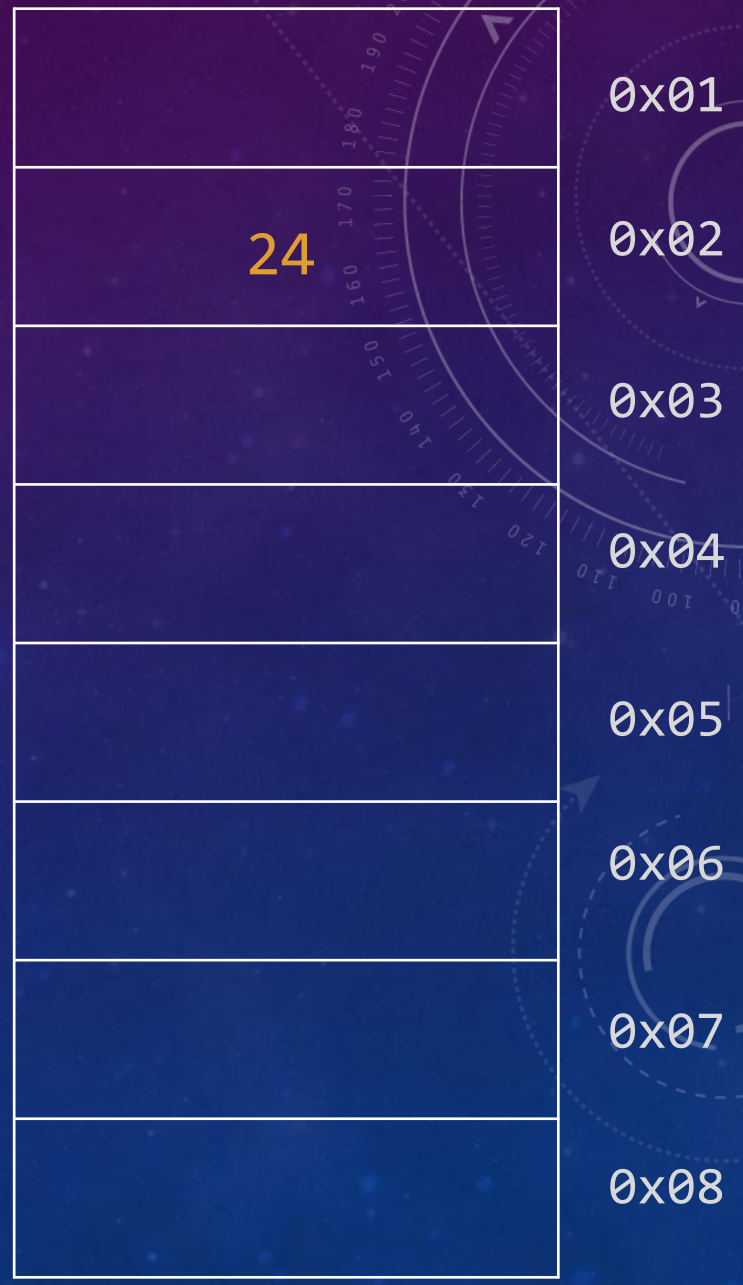
- In C++, you can create a reference to another variable using the &.



References

```
int age {24};

int & ageRef {age};
```

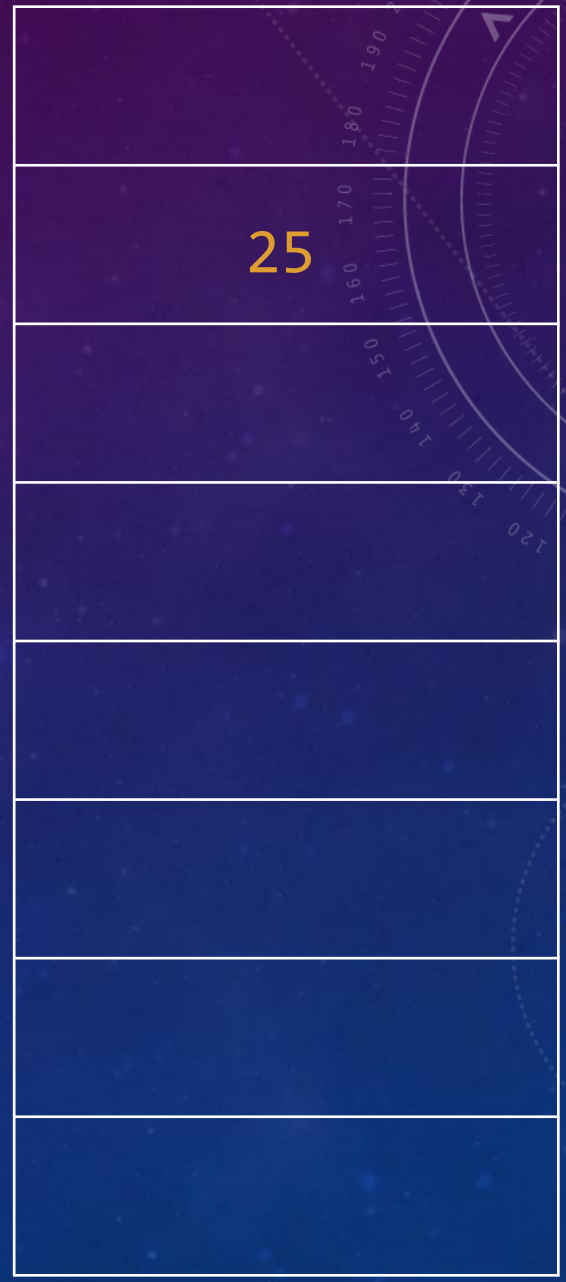
- In C++, you can create a reference to another variable using the `&`.
- Here, `ageRef` creates a reference to `age` and they both share the same memory address



References

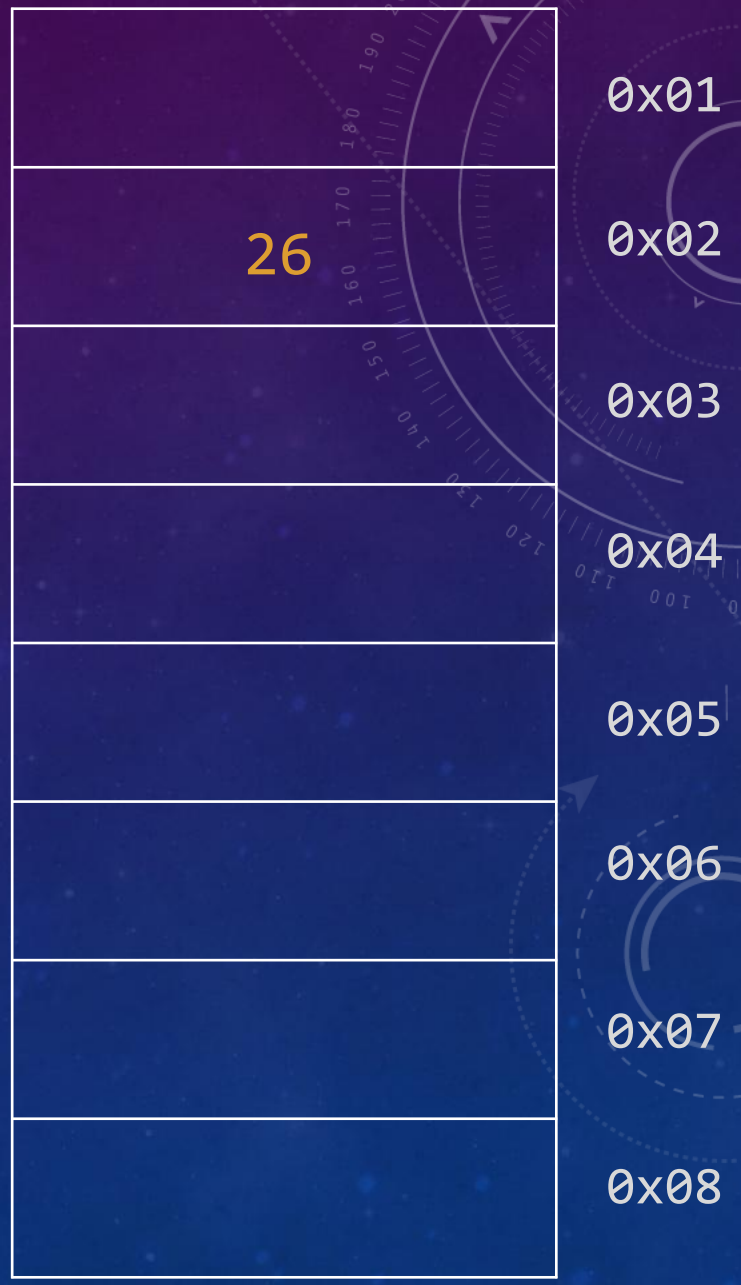
```
int age {24};

int & ageRef {age};

age = 25;
```

ageRef, age



References

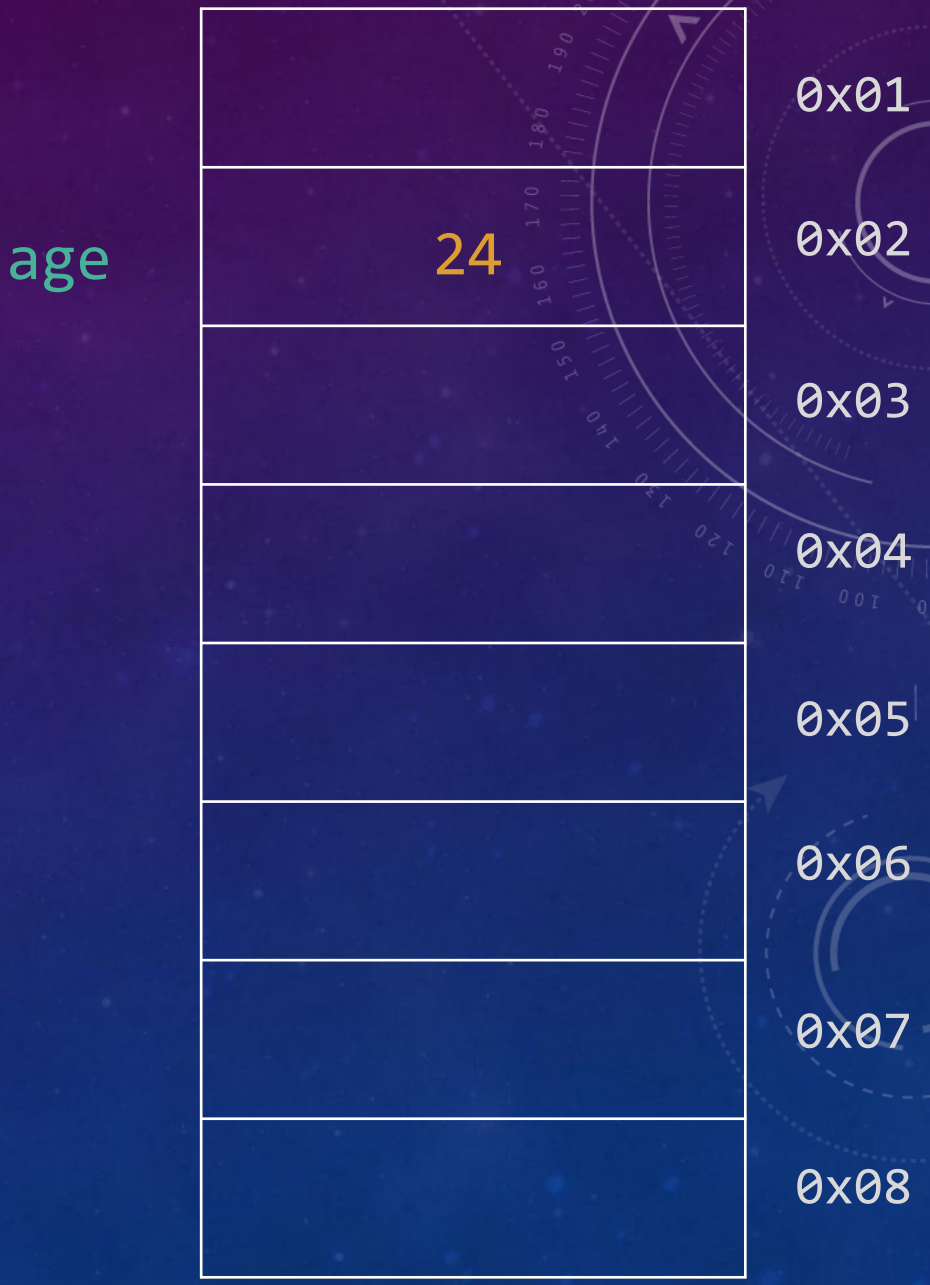
```
int age {24};

int & ageRef {age};

age = 25;

ageRef = 26;
```



Address-of-operator

```
int age {24};

&age; // returns 0x02
```

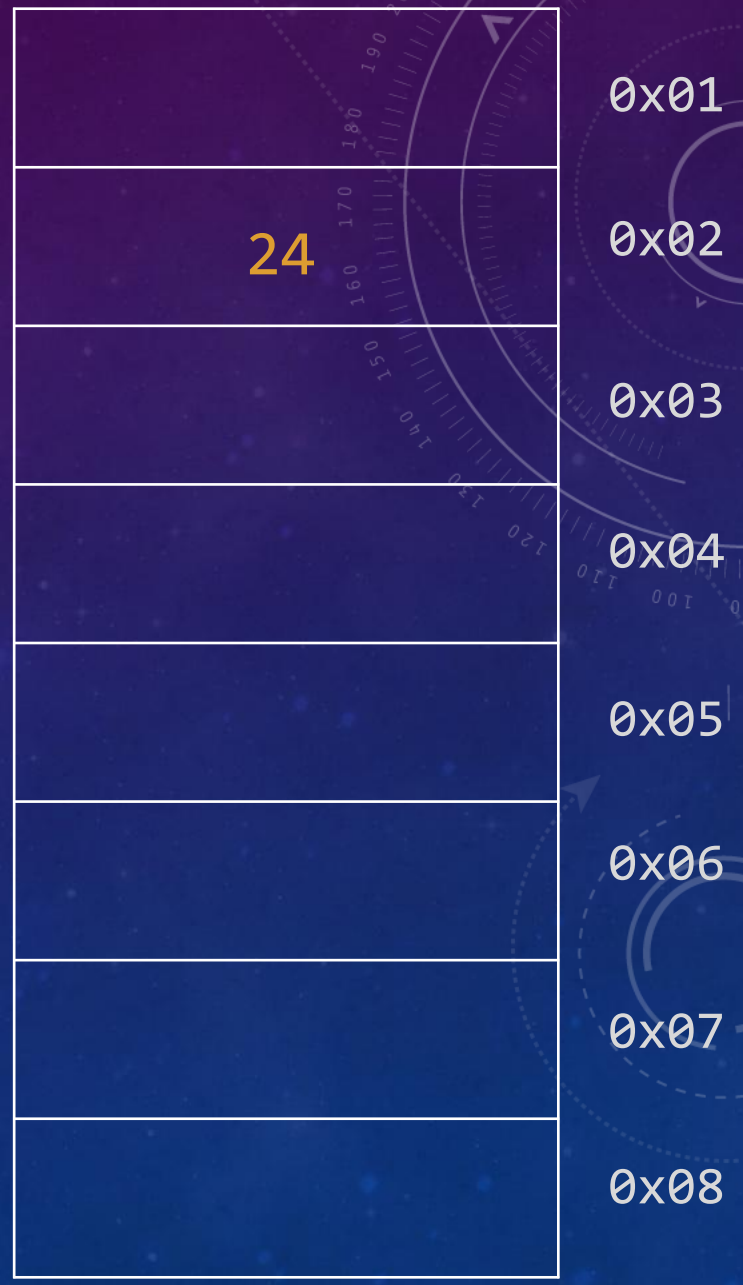
- The & can also be used to get the memory address of a variable, called either a reference declaration or an address-of-operator depending on the context



Pointers

```
int age {24};
```

- In C++, you can create a pointer to another variable's memory address using *



Pointers

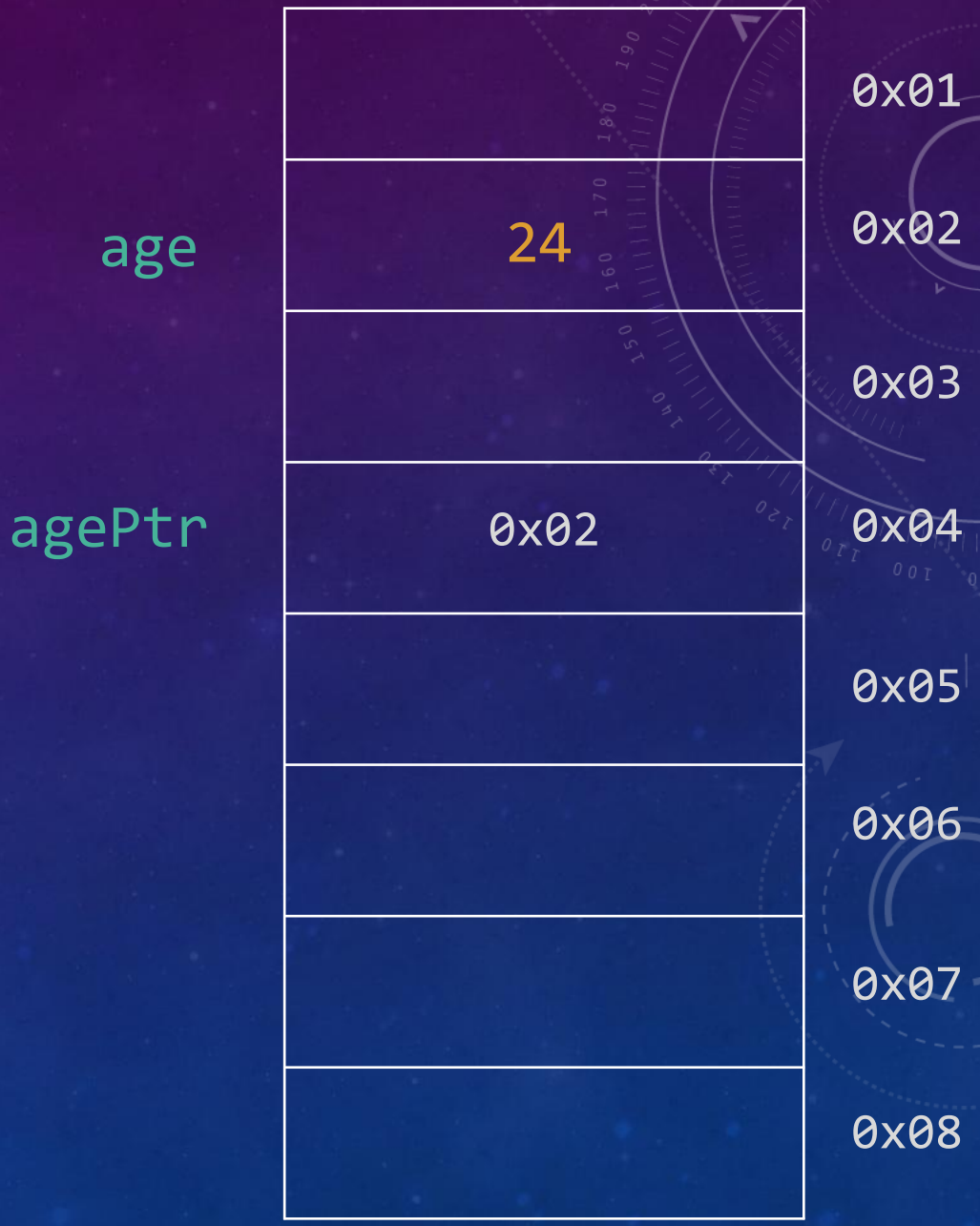
```
int age {24};

int *agePtr;
```

- In C++, you can create a pointer to another variable's memory address using *
- Different to a reference, a pointer is stored in a separate free memory address

Pointers

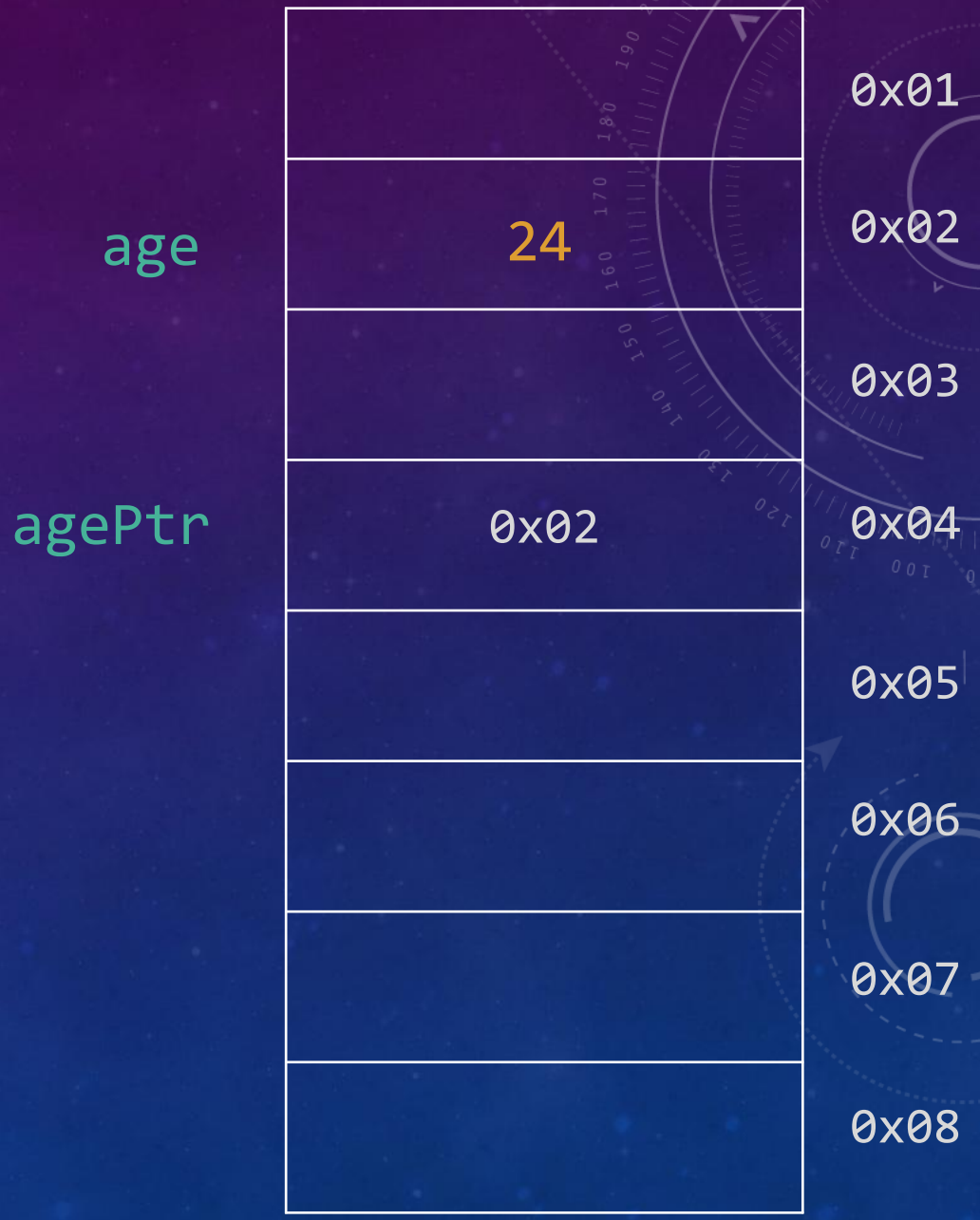
```
int age {24};

int *agePtr {&age};
```

- In C++, you can create a pointer to another variable's memory address using *
- Different to a reference, a pointer is stored in a separate free memory address
- We can then use this pointer to set the value to the memory address of age



Pointer dereferencing

```
int age {24};

int *agePtr {&age};

agePtr; // returns 0x02
```



- The * is also called the “dereference operator”, which will dereference the pointer and access the value it is pointing to

Pointer dereferencing

```
int age {24};

int *agePtr {&age};

agePtr; // returns 0x02

*agePtr; // returns 24
```

- The * is also called the “dereference operator”, which will dereference the pointer and access the value it is pointing to

Pass by value

```
void passByValue(int a) {
   a = a * 2
};
```

- When your function is passed by value, you are creating a copy of the original value
- Any alterations you make to the value within the scope of the function, will not change the original value
- Both the original and the copy are two separate values, with two separate memory addresses

Pass by reference

```
void passByReference(int &a) {
    a = a * 2
};
```

- When your function is passed by reference, the function will receive a reference to the memory address of the original value
- Meaning that any alterations you make to the value within the scope of the function will change the original value

Pass by pointer

```
void passByPointer(int *a) {
    *a = (*a) * 2
};
```

- Similar to a pass by reference, except instead of accepting a reference in its parameters, it will receive a pointer
- Any alterations you make to the value within the scope of the function will change the original value
- As you can see, to alter the value the pointer is pointing to, you have to dereference the pointer, resulting in more code in comparison to pass by reference

Pass by const reference

```
void passByConstPointer(const int &a) {
    a = a * 2 // throws compilation error
};
```

- Similar to a pass by reference, except instead of accepting a reference in its parameters, it will receive a `const` reference
- `const` stands for “constant”, meaning that the value is a read-only value and cannot be altered
- Trying to alter a `const` value will result in a compilation error

Resources

- cppreference.com is a great tool to learn specific aspects of the C++ language
  - https://en.cppreference.com/w/
- LearnCPP.com is a great tool to learn C++ chapter by chapter, covering the basics such as variables and functions, to advanced topics like structs, classes, and operator overloading
  - https://en.cppreference.com/w/
- Leetcode.com is a great tool for learning algorithms using data structures you will learn throughout DSA, it will also help you refine on your C++
  - https://leetcode.com/
- WilliamFiset on Youtube has a large assortment of videos on understanding data structures and algorithms
  - https://www.youtube.com/@WilliamFiset-videos

Access to the google drive

- I will upload slides to the Google Drive after every class
- https://drive.google.com/drive/folders/1H5psebndM_YVyoJE-BJ_ODNJOfgq9-uI