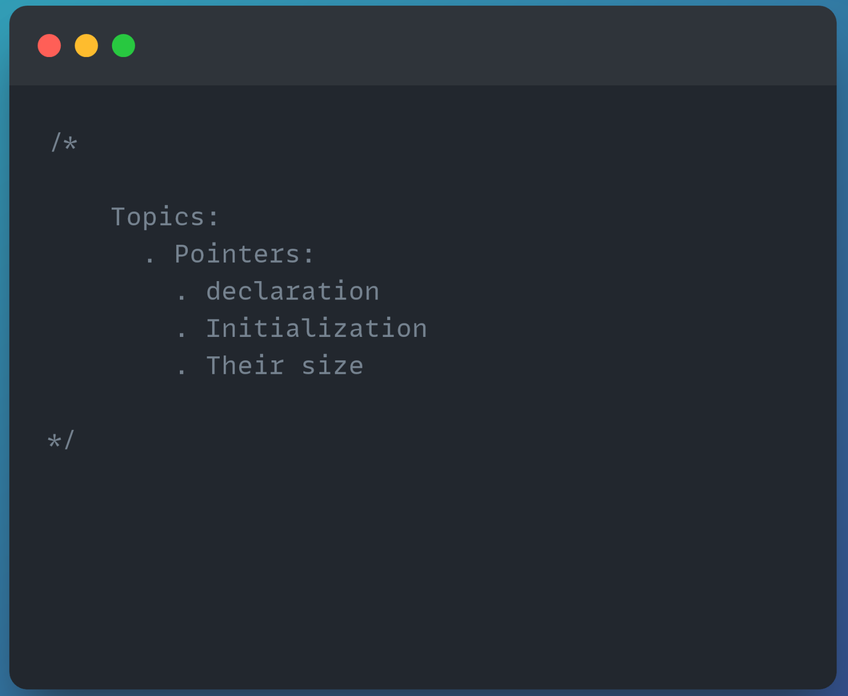
Pointers: Declaration and initialization



```
/*

    Topics:
      . Pointers:
        . declaration
        . Initialization
        . Their size

*/
```

IMPORTANT	*Raw pointers should be used sparingly, after careful consideration in modern C++. Smart pointers and RAII should be your default way of doing things, so you rarely have to deal with memory manually.

Pointers

```cpp
// Declare and initialize pointer
int *p_number{};// Will initialize with nullptr
double *p_fractional_number{};

// Explicitly initialize with nullptr
int *p_number1{ nullptr };
int *p_fractional_number1{ nullptr };

// Pointers to different variables are  of the same size
fmt::println("sizeof(int): {}", sizeof(int));// 4
fmt::println("sizeof(double): {}", sizeof(double));// 8
fmt::println("sizeof(double*): {}", sizeof(double *));
fmt::println("sizeof(int*): {}", sizeof(int *));
fmt::println("sizeof(p_number1): {}", sizeof(p_number1));
fmt::println("sizeof(p_fractional_number1): {}", sizeof(p_fractional_number1));
```

Pointers

```cpp
// Declare and initialize pointer
int *p_number{};// Will initialize with nullptr
double *p_fractional_number{};

// Explicitly initialize with nullptr
int *p_number1{ nullptr };
int *p_fractional_number1{ nullptr };

// Pointers to different variables are  of the same size
fmt::println("sizeof(int): {}", sizeof(int));// 4
fmt::println("sizeof(double): {}", sizeof(double));// 8
fmt::println("sizeof(double*): {}", sizeof(double *));
fmt::println("sizeof(int*): {}", sizeof(int *));
fmt::println("sizeof(p_number1): {}", sizeof(p_number1));
fmt::println("sizeof(p_fractional_number1): {}", sizeof(p_fractional_number1));
```

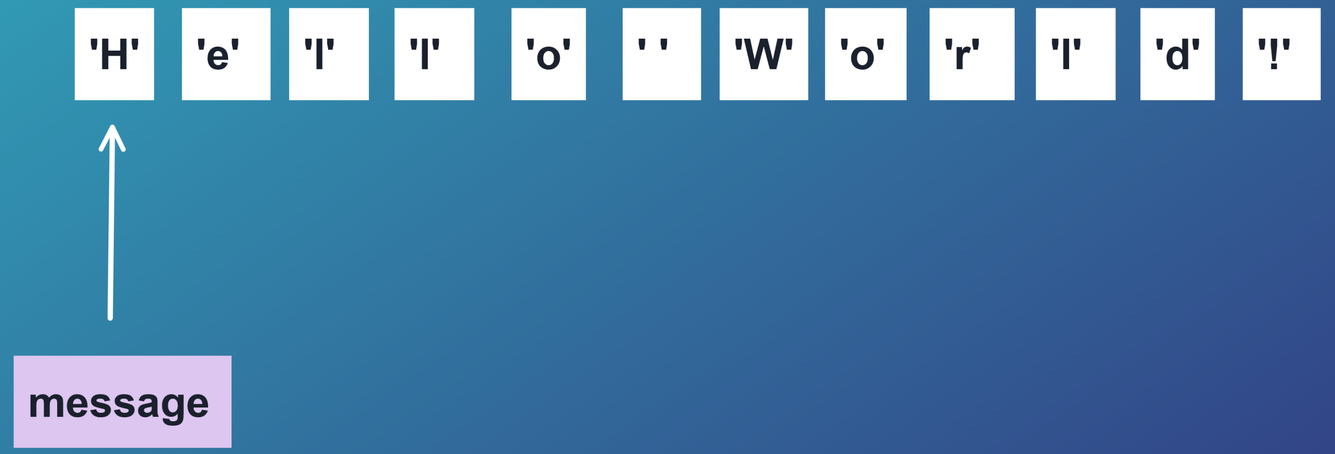
Pointers

```
// It doesn't matter if you put the * close to data type or to varible name
int *p_number2{ nullptr };
int *p_number3{ nullptr };
int *p_number4{ nullptr };

int *p_number5{}, other_int_var{};
int *p_number6{}, other_int_var6{};// Confusing as you wonder if other_int_var6
                  // is also a pointer to int. It is not
                  // The structure is exactly the same for the
                  // previous statement

fmt::println("sizeof(p_number5): {}", sizeof(p_number5));
fmt::println("sizeof(other_int_var): {}", sizeof(other_int_var));
fmt::println("sizeof(p_number6): {}", sizeof(p_number6));
fmt::println("sizeof(other_int_var6): {}", sizeof(other_int_var6));

// It is better to separate these declarations on different lines though
int *p_number7{};
int other_int_var7{};// No room for confusion this time
```

Pointers

```cpp
// Initializing pointers and assigning them data
// We know that pointers store addresses of variables
int int_var{ 43 };
int *p_int{ &int_var };// The address of operator (&);

fmt::println("Int var: {}", int_var);
fmt::println("p_int (Address in memory): {}", fmt::ptr(p_int));

// You can also change the address stored in a pointer any time
int int_var1{ 65 };
p_int = &int_var1;// Assign a different address to the pointer
fmt::println("p_int (with different address): {}", fmt::ptr(p_int));


// Can't cross assign between pointers of different types
int *p_int1{ nullptr };
double double_var{ 33 };

// p_int = &double_var; // Compiler error

// Dereferencing a pointer : {}
int *p_int2{ nullptr };
int int_data{ 56 };
p_int2 = &int_data;

fmt::println("value: {}", *p_int2);// Dereferencing a pointer
```

Pointers to char



Pointers to char

```cpp
//2.Pointers to char
 const char *message{ "Hello World!" };
 fmt::println("message: {}", message);  //This works
 fmt::println("message: {}", *message);  //This also works

 //Can't modify a through a pointer to a string literal
 //*message = "B"; // Compiler error
 fmt::println("*message: {}", *message);

 //Printing the address of the first character
 fmt::println("message address: {}", fmt::ptr(message));

 // Allow users to modify the string
 char message1[]{ "Hello World!" };
 message1[0] = 'B';
 fmt::println("message1: {}", message1);
```

const pointers and pointer to const

```
/*
            Const pointer and pointer to const
                  . Topics:
                        . Raw variables that can be modified
                              . Their addresses
                        . Non const pointer to non const data
                              . Both the data and the pointer can change
                        . Non const pointer to const data
                              . The pointer can change
                              . The data can't change(through the pointer)
                        . Const pointer to const data
                              . The pointer can't change
                              . The data can't change
                        . Const pointer to non const cata
                        . The syntax can be brutal but we'll survive.😊

*/
```

const pointers and pointer to const

```
//A raw variable that can be modified
fmt::println( "Raw variable that can be modified: " );

int number {5};// Not constant, can change number any way we want
fmt::println( "number: {}", number );
fmt::println( "&number: {}" , fmt::ptr(&number));
//Modify
number = 12;
number += 7;

//Access - Print out
fmt::println( "number: {}", number );
fmt::println( "&number: {}" , fmt::ptr(&number));
```

const pointers and pointer to const

```cpp
// Pointer: Can modify the data and the pointer itself
int *p_number1 {nullptr};
int  number1{23};

p_number1 = &number1;
fmt::println( "Pointer and value pointed to: both modifiable." );
fmt::println( "p_number1: {}" , fmt::ptr(p_number1)); // Address
fmt::println( "*p_number1: {}" , *p_number1 ); // 23
fmt::println( "number1: {}" , number1 ); // 23

//Change the pointed to value through pointer
fmt::println( "Modifying the value pointed to p_number1 through the pointer: " );
*p_number1 = 432;
fmt::println( "p_number1: {}" , fmt::ptr(p_number1));
fmt::println( "*p_number1: {}" , *p_number1 );
fmt::println( "number1: {}" , number1 );

//Change the pointer itself to make it point somewhere else
fmt::println( "Changing the pointer itself to make it point somewhere else" );
int number2 {56};
p_number1 = &number2;
fmt::println( "p_number1: {}" , fmt::ptr(p_number1));
fmt::println( "*p_number1: {}" , *p_number1 );
fmt::println( "number1: {}" , number1 );
fmt::println( "number2: {}" , number2 );
```

const pointers and pointer to const

```cpp
// Pointer to const
// Pointer pointing to constant data : You can't modify the data through pointer
fmt::println( "Pointer is modifiable, pointed to value is constant: " );
int number3 {632};  // Although you can omit the const on number3 here and it is still
                    // going to compile, it is advised to be as explicit as possible in
                    // your code and put the const in front. Make your intents CLEAR.

const int* p_number3 {&number3}; // Can't modify number3 through p_number3

fmt::println( "p_number3: {}" , fmt::ptr(p_number3));
fmt::println( "*p_number3: {}" , *p_number3 );

fmt::println( "Modifying the value pointed to by p_number3 through the pointer (Compile Error): " );
//*p_number3 = 444; // Compile error

//Although we can't change what's pointed to by p_number3,
    //we can still change where it's pointing

fmt::println( "Changing the address pointed to by p_number3: " );
int number4 {872};
p_number3 = &number4;

fmt::println( "p_number3: {}" , fmt::ptr(p_number3));
fmt::println( "*p_number3: {}" , *p_number3 );
```

const pointers and pointer to const

```cpp
// const keyword applies to the variable name.
fmt::println( "const keyword applies to a variable name: ");

int protected_var {10}; // Can make it const to protect it if we want.

//p_protected_var is a pointer to const data, we can't
//modify the data through this pointer: regardless of
//whether the data itself is declared const or not.
const int* p_protected_var {&protected_var};

//*p_protected_var = 55;
protected_var = 66;
fmt::println( "protected_var : {}" , protected_var);
fmt::println( "p_protected_var : {}" , fmt::ptr(p_protected_var));
fmt::println( "*p_protected_var : {}" , *p_protected_var);

//You can't set up a modifiable pointer to const data though,
//You'll get a compiler error :Invalid convertion from 'const type*' to 'type*'.
const int some_data{55};
//int * p_some_data {&some_data}; // Compiler error.
//*p_some_data = 66;
```

const pointers and pointer to const

```
// Both pointer and pointed to value are constant
const int number5 {459};
const int* const p_number5 {&number5 };
fmt::println( "Pointer is constant, value pointed to is constant: " );
fmt::println( "p_number5: {}" , fmt::ptr(p_number5));
fmt::println( "*p_number5: {}" , *p_number5);

//Can't modify the pointed to value through the pointer
fmt::println( "Changing value pointed to by p_number5 through the pointer (Compile Error) " );
//*p_number5 = 222; // Error : Trying to assign to read only location

//Can't change where the pointer is pointing to: The pointer is now itself constant
fmt::println( "Changing the pointer p_number5 address itself (Compile Error) ");
int number6 {333};
//p_number5 = &number6;  // Error : Trying to assign to read only location
```

Pointers and arrays

```
/*

            . Topics:
                  . Relationship between pointers and arrays
                  . Swapping addresses

*/
```

Pointers and arrays

```cpp
// Pointers and arrays: The relationship
// The name of the array can be used as a pointer to the first element of the array
int scores[10]{ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
int *p_score{ scores };

// Print the address
fmt::println("scores: {}", fmt::ptr(scores));// Array
fmt::println("p_score: {}", fmt::ptr(p_score));// Pointer
fmt::println("&scores[0]: {} ", fmt::ptr(&scores[0]));

// Print the content at that address
fmt::println("Printing out data at array address : ");
fmt::println("*scores: {}", *scores);
fmt::println("scores[0]: {}", scores[0]);
fmt::println("*p_score: {}", *p_score);
fmt::println("p_score[0]: {}", p_score[0]);


// Differences
int number{ 21 };
p_score = &number;

// scores = &number; // The array name is a pointer, but a special pointer that kind of identifies
//  the entire array. You'll get the error  : incompatible types in assignment
// of 'int*' to 'int[10]'

fmt::println("p_score: {}", fmt::ptr(p_score));// Pointer

// std::size() doesn't work for raw pointers
fmt::println("size: {}", std::size(scores));
//fmt::println( "size: {}" , std::size(p_score) ); // Compiler error.
```

Swapping addresses

```
int arr0[5]{ 1, 2, 3, 4, 5 };
int arr1[5]{ 6, 7, 8, 9, 10 };
int *p_arr1{ arr1 };
int *p_arr0{ arr0 };

/*
int * temp{nullptr};

temp = arr1;
arr1 = arr0;    //Can't assign to an array this way.
*/

int *temp{ nullptr };

temp = p_arr1;
p_arr1 = p_arr0;
p_arr0 = temp;
```

const pointers and pointer to const

```cpp
// Pointer is constant (can't make it point anywere else)
// but pointed to value can change
fmt::println("Pointer is contant, pointed to value can change : ");
int number7{ 982 };

int *const p_number7{ &number7 };

fmt::println("p_number7 : {}", fmt::ptr(p_number7));
fmt::println("*p_number7 : {}", *p_number7);
fmt::println("Changing value pointed to through p_number7 pointer : ");

*p_number7 = 456;
fmt::println("The value pointed to by p_number7 now is : {}", *p_number7);

int number8{ 2928 };
fmt::println("Changing the address where p_number7 is pointing (Compile Error).");
// p_number7 = &number8;
```

Pointer arithmetic

```
/*
            . Topics:
                    . Navigation
                    . Distance
                    . Comparison


*/
```

Pointer arithmetic

```cpp
// Navigation: Increment/ decrement
int scores[10]{ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

// scores++;

int *p_score{ scores };
fmt::println("Values in scores array (p_score pointer increment): ");
fmt::println("Address: {},  value: {}", fmt::ptr(p_score), *(p_score));


++p_score;// Moves froward by sizeof(int): 4 bytes
fmt::println("Address: {},  value: {}", fmt::ptr(p_score), *(p_score));

++p_score;// Moves froward by sizeof(int): 4 bytes
fmt::println("Address: {},  value: {}", fmt::ptr(p_score), *(p_score));

++p_score;// Moves froward by sizeof(int): 4 bytes
fmt::println("Address: {},  value: {}", fmt::ptr(p_score), *(p_score));

++p_score;// Moves froward by sizeof(int): 4 bytes
fmt::println("Address: {},  value: {}", fmt::ptr(p_score), *(p_score));

++p_score;// Moves froward by sizeof(int): 4 bytes
fmt::println("Address: {},  value: {}", fmt::ptr(p_score), *(p_score));
...

//Be careful about moving beyond the bounds of the array.
++p_score;// Moves froward by sizeof(int) : 4 bytes
```

Pointer arithmetic

```cpp
// Navigation: Increment/ decrement
int scores[10]{ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

int *p_score{ scores };
//Explicitly adding the integer
p_score = scores; // Reset the pointer to the start of the array
fmt::println( "scores[4] : {}" , *(p_score + 4) );  // Moves forward by 4 * sizeof(int) : 16 bytes
                                                    // Prints out 15
                                                    // Can use loops to print these things out: easier
//Can do pointer arithmetic in a loop
p_score = scores;
fmt::println( "Pointer arithmetic on fmt::ptr(p_score)s pointer and a for loop: " );
for ( size_t i{0} ; i < std::size(scores) ; ++i){
    fmt::println("Value: {}", *(p_score + i) ); // scores[i]
}

// Can also do arithmetic on the array name
// just like any pointer.
p_score = scores;

fmt::println( "Pointer arithmetic on array name: " );
for ( size_t i{0} ; i < std::size(scores) ; ++i){
    fmt::println("Value: {}", *(scores + i) );
}
```

Pointer arithmetic

```cpp
//If you subtract one pointer from another, you get a distance
int scores[10]{ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

// Array index notation
//     int * pointer1 {&scores[0]};
//     int * pointer2 {&scores[8]};

// Pointer arithmetic notation
int *pointer1{ scores + 0 };
int *pointer2{ scores + 8 };

fmt::println("pointer2 - pointer1 : {}", pointer2 - pointer1);// 8
fmt::println("pointer1 - pointer2 : {}", pointer1 - pointer2);// -8


std::ptrdiff_t pos_diff = pointer2 - pointer1;
std::ptrdiff_t neg_diff = pointer1 - pointer2;
fmt::println("pointer2 - pointer1 : {}", pos_diff);// 8
fmt::println("pointer1 - pointer2 : {}", neg_diff);// -8
fmt::println("sizeof(std::ptrdiff_t) : {}", sizeof(std::ptrdiff_t));
```

Pointer arithmetic

```
//Comparison operations
int scores[10]{ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };

int *pointer1{ &scores[0] };
int *pointer2{ &scores[8] };

// Can also compare pointers

// The further you go in the array, the bigger the address

fmt::println("Comparing pointers : ");

fmt::println("pointer1 > pointer2 : {}", (pointer1 > pointer2));
fmt::println("pointer1 < pointer2 : {}", (pointer1 < pointer2));
fmt::println("pointer1 ⩾ pointer2 : {}", (pointer1 ⩾ pointer2));
fmt::println("pointer1 ⩽ pointer2: {}", (pointer1 ⩽ pointer2));
fmt::println("pointer1 == pointer2 : {}", (pointer1 == pointer2));
fmt::println("pointer1 ≠ pointer2 : {}", (pointer1 ≠ pointer2));
```