Escape Sequences in C++ and C

	Name	Explanation
\n	Newline Line Feed	Position the screen cursor to the beginning of the next line
\t	Horizontal Tab	Move the screen cursor to the next tab stop
\r	Carriage Return	Position the screen cursor to the beginning of the current line; do not advance to the next line
\a	Alert	Sound the system bell
\\	Backslash	Used to print a backslash character
\'	Single Quote	Used to print a single-quote character
\"	Double Quote	Used to print a double-quote character

Creating own namespaces

```
// Name spaces
#include <iostream>

namespace English {
    void greeting() { std::cout << "Hello" << std::endl; }
}

namespace German {
    void greeting() { std::cout << "Guten Tag" << std::endl; }
}

int main(void) {
    English::greeting();
    German::greeting();
    return 0;
}
```

```
Hello
Guten Tag
```

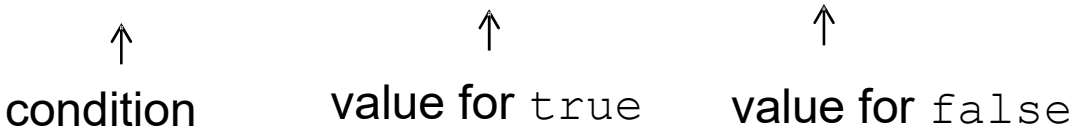
Alternative: The „?“ Operator

It’s a so called “Trenary Operator “. Takes three operands

1. Operand: condition
2. Operand: value for the conditional expression if condition `true`
3. Operand: value for the conditional expression if condition `false`

Returns the 2nd Operand if the condition is true
Returns the 3rd Operand if the condition is false

```
cout << ( score >= 60 ? "Passed" : "Failed" );
```



```
for (int year = 0; year <= 10; year++) {
     // calculate amount for the specified year
     final_amount = start_amount * std::pow(1.0 + rate / 100, year);

     // display the year and the amount
     cout << 2024 + year << std::setw(11) << final_amount << endl;
}
```

```
Year      Amount
2024      900.00
2025      937.80
2026      977.19
2027     1018.23
2028     1061.00
2029     1105.56
2030     1151.99
2031     1200.37
2032     1250.79
2033     1303.32
2034     1358.06
```

`std::setw( 11 )`

- only relates to the next output (non-sticky)
- specifies that the next value output should appear in a field width of 11 (default: right-aligned)
- Stream manipulator `std::left` changes to left aligned.

thws

Things to know about Stream Manipulators

- `fixed` and `setprecision` are often used together.
- Stream manipulator `std::fixed` indicates that floating-point values should be output as fixed-point values with decimal points.
- Stream manipulator `std::setprecision` specifies the number of digits to the right of the decimal point.

number	iomanip	representation
0.0 0.0 0.0 0.0	fixed scientific hexfloat default	0.000000 0.000000e+00 0x0p+0 0
0.01 0.01 0.01 0.01	fixed scientific hexfloat default	0.010000 1.000000e-02 0x1.47ae147ae147bp-7 0.01
0.00001 0.00001 0.00001 0.00001	fixed scientific hexfloat default	0.000010 1.000000e-05 0x1.4f8b588e368f1p-17 1e-05

Selected Math Library Functions

Function	Description	Example
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 )` is 128 `pow( 9, .5 )` is 3
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 )` is 0
`sqrt( x )`	square root of x (where x is a nonnegative value)	`sqrt( 9.0 )` is 3.0
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 )` is 0

Deitel & Deitel, C++ How to Program, 7th ed.

More mathematical functions: https://en.cppreference.com/w/cpp/header/cmath

Different ways to define references (all equivalent)

```
void square(int& y) {
   y*=y;
}
```

```
void square(int &y) {
   y*=y;
}
```

```
void square(int & y) {
   y*=y;
}
```

```
void square(int&y) {
   y*=y;
}
```

A new example: Printing the distance in meter

```
int printDistance(double distance, int precision) {
   cout << fixed << setprecision(precision);
   cout << "The distance is ";
   cout << distance << " m" << endl;
}

int main() {
   printDistance(10, 2);
   printDistance(1.2, 2);
   printDistance(1.001, 3);
}
```

```
The distance is 10.00 m
The distance is 1.20 m
The distance is 1.001 m
```

Lets say: Most of the time are interested in a accuracy in the cm area!

Operator Precedence

Operators	Associativity	Type
() []	left to right	highest
++ -- static_cast< *type* >(*operand*)	left to right	unary (postfix)
++ -- + - ! & *	right to left	unary (prefix)
* / %	left to right	multiplicative
+ -	left to right	additive
<< >>	left to right	insertion/extraction
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Deitel & Deitel, C++ How to Program

Pass-by-Address

```
void swap(int* i1_ptr, int& i2_ptr) {
   int t; // temporary storage
   t = *i1_ref;
   *i1_ref = *i2_ref;
   *i2_ref = t;
}
```

```
int main() {
   int a = 1;  int b = 2;
   int* c = &b;

   swap(&a, c);
}
```

Arguments passed by value can be
- Pointer variables, e.g. c
- ~~literals (e.g. 6),~~
- Expressions returning pointers, e.g. &a

When to use pass by address?
- Advantage: Can be used in pure C, no copy of large datastructures necessary, can handle a nullprt value, arrays are also passed this way
- Disadvantage: handling with pointers can be error-prone

Processing C-Strings with Library Functions

```
char* strcpy( char* s1_ptr, const char* s2_ptr );
```

- Copies the string pointed to by `s2_ptr` into the character array `s1`. The value of `s1_ptr` is returned.

```
char* strncpy( char* s1_ptr, const char* s2_ptr, size_t n );
```

- Copies at most `n` characters of the string pointed to by `s2_ptr` into the character array pointed to by `s1_ptr`. The value of `s1_ptr` is returned.

```
char* strcat( char* s1_ptr, const char* s2_ptr );
```

- Appends the string pointed to by `s2_ptr` to the string pointed to by `s1_ptr`. The first character associated with `s2_ptr` overwrites the terminating null character of the string pointed to by `s1_ptr`. The value of `s1_ptr` is returned.

```
char* strncat( char* s1_ptr, const char* s2_ptr, size_t n );
```

- Appends at most `n` characters of string pointed to by `s2_ptr` to string pointed to by `s1_ptr`. The first character of string associated with `s2_ptr` overwrites the terminating null character of string pointed to by `s1_ptr`. The value of `s1_ptr` is returned.

Processing C-Strings with Library Functions

```
int strcmp(const char* s1_ptr, const char* s2_ptr);
```

- Compares string pointed to by `s1_ptr` with the string pointed to by `s2_ptr`. The function returns a value of zero, less than zero or greater than zero if string associated with `s1_ptr` is equal to, less than or greater than string associated with `s2_ptr`.

```
int strncmp(const char* s1_ptr, const char* s2_ptr, size_t n);
```

- Compares up to `n` characters of the string pointed to by `s1_ptr` with the string associated with `s2_ptr`. The function returns zero, less than zero or greater than zero if the ncharacter portion of string associated with `s1_ptr` is equal to, less than or greater than the corresponding n-character portion of string associated with `s2_ptr`.

```
size_t strlen( const char* s_ptr );
```

- Determines the length of string pointed to by `s_ptr`. The number of characters preceding the terminating null character is returned.

Using const with Pointers

3. Constant Pointer to non-constant Data

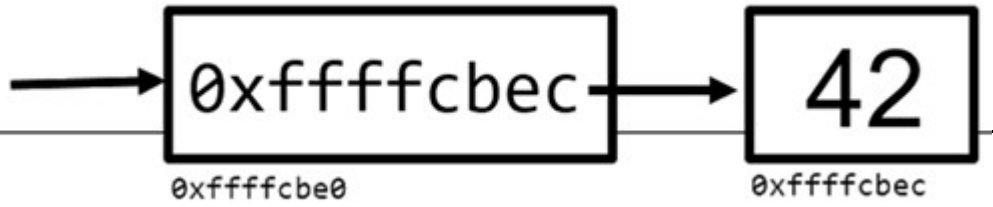
```
char* const c_ptr3 = text2;
```

- Pointer always points to the same memory location, cannot be reseated.
- Data can be changed via the pointer.
- Must be initialized during declaration
- **Conceptually very similar to references.**

4. Constant Pointer to constant Data

```
const char* const c_ptr4 = text2;
```

- Option with the fewest possible changes.
- Always points to the same memory location.
- Data cannot be changed using this pointer.



String Conversion Operators

Function	Description
std::stoi(str)	Converts string str to int, returns int
std::stol(str)	Converts string str to long, returns long
std::stoul(str)	returns unsigned long
std::stoll(str)	returns long long
std::stoull(str)	returns unsigned long long
std::stof(str)	Converts string str to float, returns float
std::stod(str)	returns double
std::stold(str)	returns long double

Header `<cstring>` must be included to use the string conversion functions.

```cpp
#include <iostream>
#include <vector>
using namespace std;

void print(const vector<int> &items);
void tryToModifyArray(vector<int> a);
void modifyArray(vector<int>& a_ref);

int main() {
    vector<int> items{ 1, 2, 3, 4, 5 };
    print(items);

    // call by value - complete array is copied
    tryToModifyArray(items);
    print(items);

    // call by reference;
    modifyArray(items);
    print(items);

    return 0;
}   // code will be continued on next slide
```

Overview of important C++ Vector commands

```
v[n];
// Represents n-th element in v, no range check, i.e., v[n] may not exist.

v.at(n);
// Represents n-th element in v, with range check,
// exception is thrown, if v[n] does not exist.

v.size();
// Returns number of vector elements already occupied (vector size)

v.capacity();
// Returns total number of elements that can be stored in v without
// memory reallocation

v.reserve(n);
// Reserves minimum amount of mememory for n elements. This memory
// can then be worked with without having to request new memory.
```

Overview of important C++ Vector commands

```
v.push_back(val);
// Adds data with value val to the end of vector v, increments size by one

v.pop_back();
// Removes last element of vector v, shrinks size by one.

v.front();
// Returns reference to first element in v.

v.back();
// Returns reference to last element in v.
```

Overview of important C++ Vector commands

```
v.begin();
// Returns iterator to first element in v (iterator is similar to pointer)

v.end();
// Returns iterator to last element in v (actually
// to the element after the last element in v)

v.erase(i);
// Removes i-th element from v.

v.insert(i, val);
// Inserts new element with value val in front of element with index i.

vcp = v;
// Makes vcp a copy of v.

v.clear();
// Removes all elements of v, empties v.
```

Important Member Functions of the STL String

The C++ class `string` provides many useful member functions.

Function	Purpose
`size()`	Returns string length
`length()`	Returns string length (synonym to `size()`)
`insert(n,s)`	Inserts string `s` at position `n`
`erase(p,n)`	Starting at position `p` removes the next `n` characters
`find(s)`	Returns position `n` at which string `s` was found.

```
int main()
{
   std::string s("123");     // Constructor involving C-String
   int len = s.length();     // returns 3, the string length
   s.insert(2, "xy");        // s is now "12xy3"
   s.erase(2, 2);            // s is now "123"
   int pos = s.find("23");   // returns position 1
}
```

Converting To Strings

The following functions can be used to convert fundamental data types to C++ `strings`.

```
#include <string>
using namespace std;
string to_string(int value);
string to_string(long value);
string to_string(long long value);
string to_string(unsigned int value);
string to_string(unsigned long value);
string to_string(unsigned long long value);
string to_string(float value);
string to_string(double value);
string to_string(long double value);
```

Sorting vectors using operator<

```
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector
using namespace std;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  vector<int> myvector (myints, myints+8);
  sort (myvector.begin(), myvector.begin()+4);

  for (int value : myvector) // print content
    cout << ' ' << value;
  cout << endl;
  return 0;
}
```

```
 12 32 45 71 26 80 53 33
```