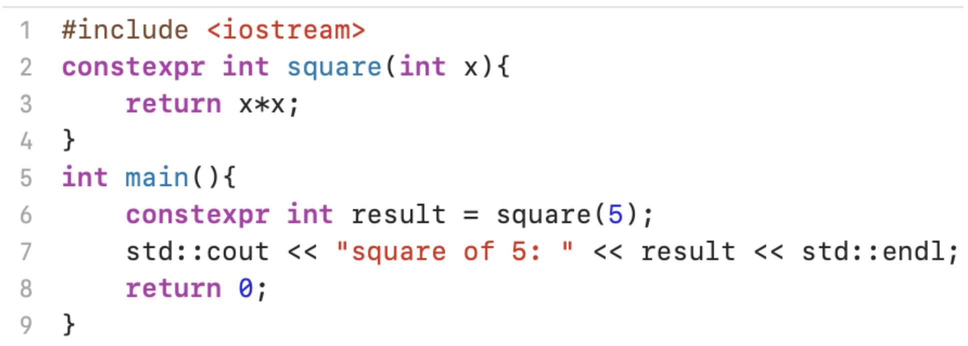
Variables:

- **const**: It is used to declare constant variable and its value cannot be modified.
- **constexpr**: It is used to declare that an expression or function can be evaluated at compile time. It's used for creating constants and performing computations during compilation. It helps your program run faster and use less memory. (To compile your program with C++11 support)

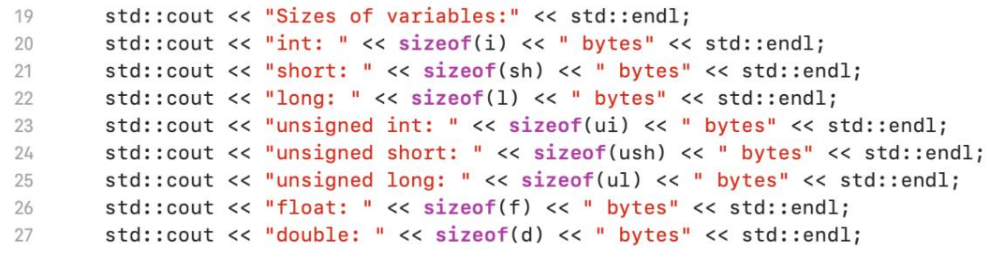
Example of use:



```cpp
#include <iostream>
constexpr int square(int x){
    return x*x;
}
int main(){
    constexpr int result = square(5);
    std::cout << "square of 5: " << result << std::endl;
    return 0;
}
```

- **sizeof**: This function declares variable’s size. For example:

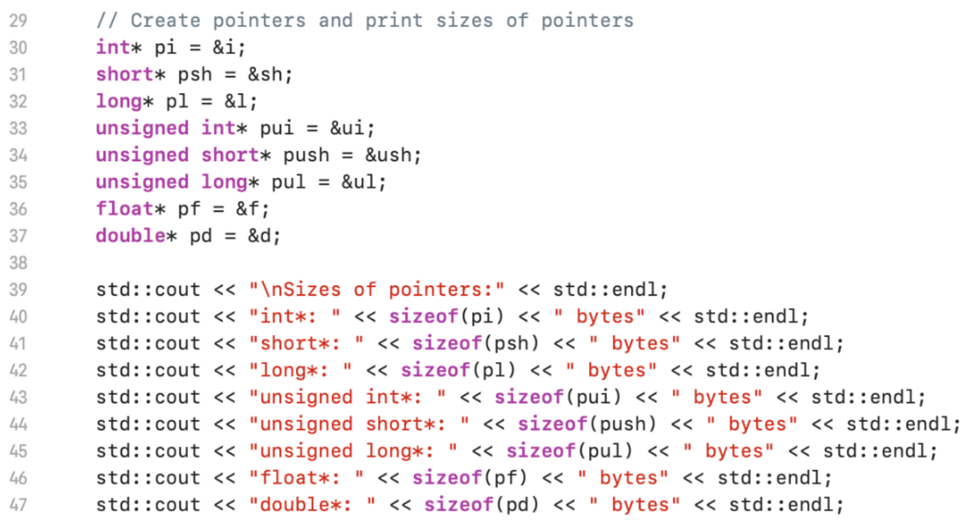
To sizeof variables, this example can be given:



```cpp
    std::cout << "Sizes of variables:" << std::endl;
    std::cout << "int: " << sizeof(i) << " bytes" << std::endl;
    std::cout << "short: " << sizeof(sh) << " bytes" << std::endl;
    std::cout << "long: " << sizeof(l) << " bytes" << std::endl;
    std::cout << "unsigned int: " << sizeof(ui) << " bytes" << std::endl;
    std::cout << "unsigned short: " << sizeof(ush) << " bytes" << std::endl;
    std::cout << "unsigned long: " << sizeof(ul) << " bytes" << std::endl;
    std::cout << "float: " << sizeof(f) << " bytes" << std::endl;
    std::cout << "double: " << sizeof(d) << " bytes" << std::endl;
```

Output: “Sizes of variables:
int: 4 bytes
short: 2 bytes
long: 8 bytes
unsigned int: 4 bytes
unsigned short: 2 bytes
unsigned long: 8 bytes
float: 4 bytes
double: 8 bytes”

By writing this codes, you can find size of variables. Also, let us look them pointer's sizes:



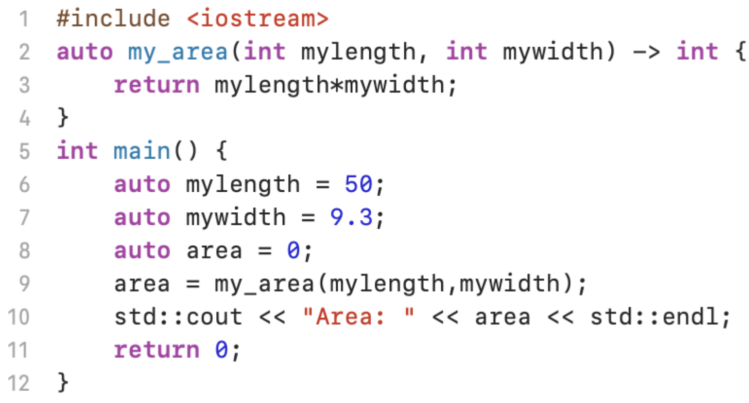
```cpp
    // Create pointers and print sizes of pointers
    int* pi = &i;
    short* psh = &sh;
    long* pl = &l;
    unsigned int* pui = &ui;
    unsigned short* push = &ush;
    unsigned long* pul = &ul;
    float* pf = &f;
    double* pd = &d;

    std::cout << "\nSizes of pointers:" << std::endl;
    std::cout << "int*: " << sizeof(pi) << " bytes" << std::endl;
    std::cout << "short*: " << sizeof(psh) << " bytes" << std::endl;
    std::cout << "long*: " << sizeof(pl) << " bytes" << std::endl;
    std::cout << "unsigned int*: " << sizeof(pui) << " bytes" << std::endl;
    std::cout << "unsigned short*: " << sizeof(push) << " bytes" << std::endl;
    std::cout << "unsigned long*: " << sizeof(pul) << " bytes" << std::endl;
    std::cout << "float*: " << sizeof(pf) << " bytes" << std::endl;
    std::cout << "double*: " << sizeof(pd) << " bytes" << std::endl;
```

Output: "Sizes of pointers:
int*: 8 bytes
short*: 8 bytes
long*: 8 bytes
unsigned int*: 8 bytes
unsigned short*: 8 bytes
unsigned long*: 8 bytes
float*: 8 bytes
double*: 8 bytes"

- Pointer variables need to be large enough to hold memory addresses, which depend on the memory address space of the computer architecture, while other variables are sized based on the representation of their data type.

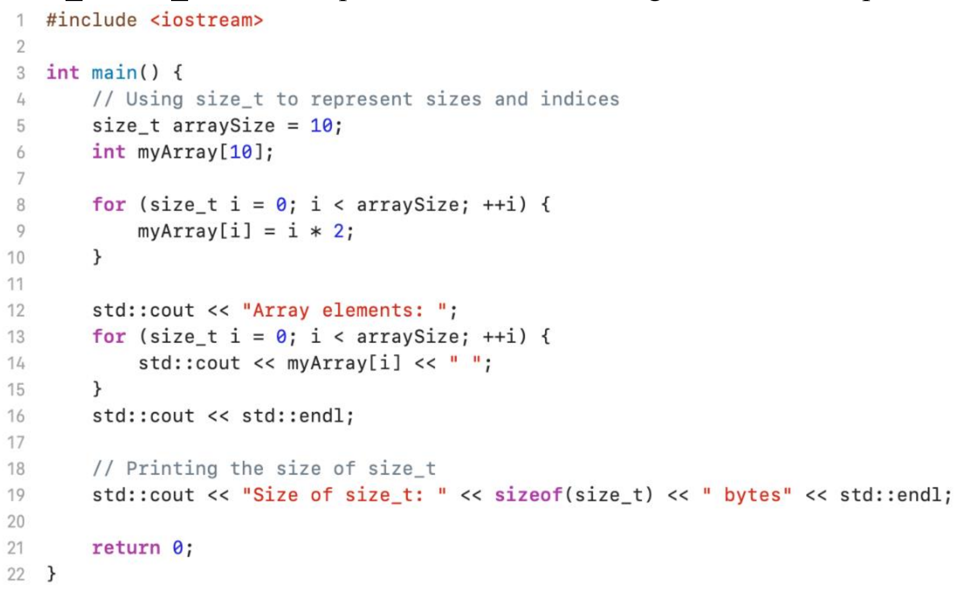
- **auto**: In C++, the '**auto**' keyword is used for automatic type inference. It allows the compiler to automatically deduce the type of a variable based on its initializer.



```cpp
#include <iostream>
auto my_area(int mylength, int mywidth) -> int {
    return mylength*mywidth;
}
int main() {
    auto mylength = 50;
    auto mywidth = 9.3;
    auto area = 0;
    area = my_area(mylength,mywidth);
    std::cout << "Area: " << area << std::endl;
    return 0;
}
```

Output: "Area: 225"

- **int**: Represents integers (whole numbers).Commonly used for counting, indexing, and general-purpose integer arithmetic
- **char**: represent single characters. Used for storing ASCII characters and small integers.
- **long**: long can be used with int, double. For example; when it is used with int, int variable's size is larger. "long long int" can be used and it has same size with "long int" on 64 bit computers. However, they have different size on 32 bit computers.
- **float**: Float uses 32 bits of memory. It has 7 digits of precision.
- **double**: Double uses 64 bits of memory. It has 15 digits of precision.1515.6
- **unsigned**: unsigned makes variables non-negative. For example, short valuables are between –32768 and 32767 but when it is unsigned short, it is between 0 and 65535.
- "size_t": size_t indicates space and it cannot be negative. For example,



```cpp
#include <iostream>

int main() {
    // Using size_t to represent sizes and indices
    size_t arraySize = 10;
    int myArray[10];

    for (size_t i = 0; i < arraySize; ++i) {
        myArray[i] = i * 2;
    }

    std::cout << "Array elements: ";
    for (size_t i = 0; i < arraySize; ++i) {
        std::cout << myArray[i] << " ";
    }
    std::cout << std::endl;

    // Printing the size of size_t
    std::cout << "Size of size_t: " << sizeof(size_t) << " bytes" << std::endl;

    return 0;
}
```

Output:

```
Array elements: 0 2 4 6 8 10 12 14 16 18
Size of size_t: 8 bytes
```

- **int(N)_t and float(N)_t:** The (N)_t types, such as int32_t, int64_t, float32_t, float64_t, etc., are fixed-width integer and floating-point types defined in C++ programming languages. These types are part of the **<cstdint>** in C++.

-"int(N)_t": These types ensure a specific size (in bits) regardless of the platform. For example, **int32_t** guarantees a 32-bit signed integer.
-"float(N)_t": Similar to int(N)_t, these types guarantee a specific size for floating-point numbers. For instance, **float64_t** ensures a 64-bit double-precision floating-point number.

- “**char8_t**”: Introduced in C++20 to support UTF-8 character literals. It's an 8-bit character type. Used for representing UTF-8 encoded characters.
- **“char16_t”:** A 16-bit character type. Useful for representing UTF-16 encoded characters. Can be used to store Unicode characters.
- **“char32_t”:** A 32-bit character type. Used for representing UTF-32 encoded characters. Suitable for storing any Unicode character.
- **“wchar_t”:** A wide character type whose size is platform-dependent. Used for representing wide characters, especially in functions that support wide characters (e.g., **wcout** in C++ for wide character output).