sizeof

Xavier Bonaventura

Topalsson GmbH

Definition

- sizeof(*type*)
  - Returns size in bytes of the object representation of *type*.

- How many bits a byte consist of according to the standard?
  - Depends on the architecture
  - At least 8
  - CHAR_BIT

Basic types

	Standard	C++ Shell (GCC 4.9.2)
	`sizeof(foo)`	`sizeof(foo)`
`char foo;`	1	1
`bool foo;`	Implementation-defined	1
`short foo;`	Implementation-defined	2
`int foo;`	Implementation-defined	4
`long foo;`	Implementation-defined	8
`int* foo;`	Implementation-defined	8
`float foo;`	Implementation-defined	4
`double foo;`	Implementation-defined	8
`long double foo;`	Implementation-defined	16

But the standard guarantee:

```
1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
```

sizeof(long long) could be 1!

Xavier Bonaventura - Topalsson GmbH

Common data models

- 32 bit systems:
  - **LP32** or **2/4/4** (int is 16-bit, long and pointer are 32-bit)
    - Win16 API
  - **ILP32** or **4/4/4** (int, long, and pointer are 32-bit);
    - Win32 API
    - Most Unix and Unix-like systems (Linux, Mac OS X)
- 64 bit systems:
  - **LLP64** or **4/4/8** (int and long are 32-bit, pointer is 64-bit)
    - Win64 API
  - **LP64** or **4/8/8** (int is 32-bit, long and pointer are 64-bit)
    - Most Unix and Unix-like systems (Linux, Mac OS X)

Class and structs

	C++ Shell (GCC 4.9.2) `sizeof(foo)`	

`struct my_struct`
`{`
`} foo;`
	1	Empty class allways 1

`struct my_struct`
`{`
`  bool a;`
`} foo;`
	1	Zero overhead

`struct my_struct`
`{`
`  int a;`
`} foo;`
	4	Zero overhead

Class and structs

```
struct my_struct
{
    bool a;
    int b;
} foo;
```

8

```
bool a;
char pad[3];
int b;
```

```
struct my_struct
{
    bool a;
    int b;
    bool c;
} foo;
```

12

```
bool a;
char pad1[3];
int b;
bool c;
char pad2[3];
```

```
struct my_struct
{
    bool a;
    bool b;
    int c;
} foo;
```

8

```
bool a;
bool b;
char pad1[2];
int c;
```

References and arrays

- References

```
T &foo = bar;
```

```
sizeof(foo) == sizeof(T)
```

- Built-in

```
T foo[N];
```

```
sizeof(foo) == sizeof(T)*N
```

- std::array

```
std::array<T, N> foo;
```

```
sizeof(foo) == sizeof(T)*N
```

Vectors

`std::vector<bool> foo(10,0);`	**C++ Shell (GCC 4.9.2)**
`sizeof(foo)`	~~10~~ 40
`sizeof(foo[0])*foo.size()`	~~10~~ 160
`typeid(foo[0]).name() != bool`	

```
size_t my_sizeof(const std::vector<T>& vec)
{
        return sizeof(T)*vec.size();
}
```

Thank you!