C/C++: Lecture 2

Vorobev D.V

11.09.2020

Memory

Cell

Facts:

- It is a minimal addressing memory unit
- It has an address
- It is a bounded linear sequence of bits
- Through out the history of computers design the sizes of cells were different
- Nowadays the size of the cell equals 8 bits

Address

Facts:

- It is a natural number that enumerates the fixed cell
- The size of the set of natural numbers is determined by the computer architecture **bit width**.
- For the 32 bit width arch the set of natural numbers is $\{0, 1, 2...2^{32} - 1\}$

Byte

Facts:

- 8 bit width cell is called byte

Thus:

- 1. On the "32 bit width architecture" there are 2^{32} cells encoded
- 2. 1 cell = 1 byte
- 3. From the points **1** and **2** on the "32 bit width arch" the size of the virtual memory is 2^{32} bytes $\Leftrightarrow$ 4GB

Word

Facts:

- Is a bounded sequence of bytes
- For the ''32 bit width arch'' the size of the word equals 4 bytes
- For the ''32 bit width arch'' the size of the word equals 8 bytes

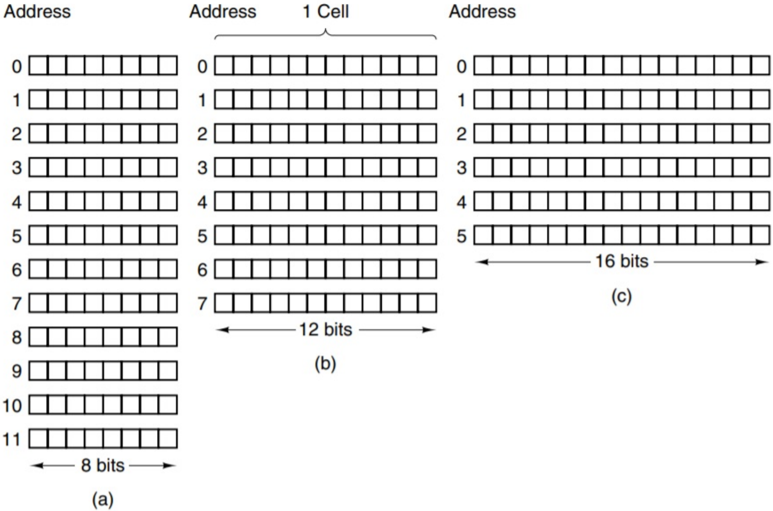
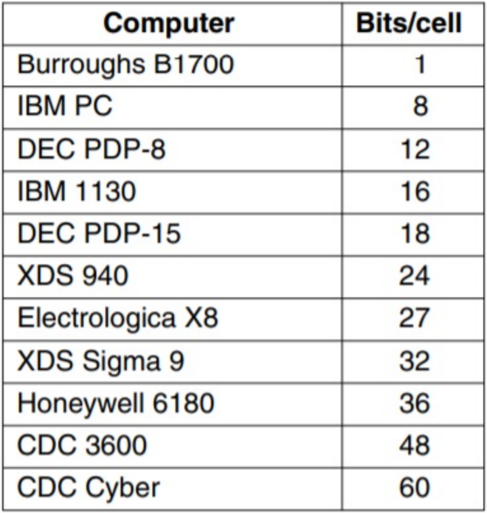


Figure: Variants of the cell size



Computer	Bits/cell
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Figure: Cell sizes in different machines

Endian

Big endian

It is the mapping of **the most** significant byte **to the lowest address** in the virtual memory.

Little endian

It is the mapping of **the least** significant byte **to the lowest address** in the virtual memory.

Endian:Example

Let's consider a variable int32_t x = 1614 and 32-bit width architecture. Thus:

1. the word size is 4 bytes
2. $x = 0000011001001110_2$ - the binary representation
3. the **most** significant byte is **0000**
4. the **least** significant byte is **1110**

Let's x maps to the range of 4 addresses:

1. 0xFFFFBA67CC980011
2. ...
4. 0xFFFFBA67CC980000

Here 0xFFFFBA67CC980000 is the lowest address.
On the **big** endian arch byte **0000** maps to the 0xFFFFBA67CC980011.
On the **little** endian arch byte **1110** maps to the 0xFFFFBA67CC980011.

Operators

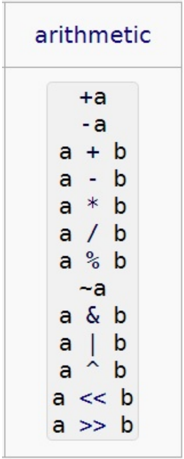
Precedence

It is a property that determines the order of calling operator.

Associativity

It is a property that determines the order of placing brackets.

Arithmetic



arithmetic
+a
-a
a + b
a - b
a * b
a / b
a % b
~a
a & b
a | b
a ^ b
a << b
a >> b

Arithmetic

Right associative

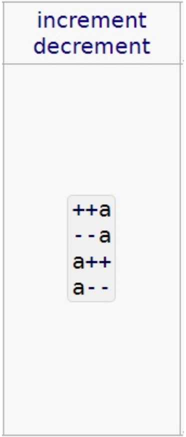
+ (unary) - (unary)

Left associative

+ (binary) - (binary) * / % & | ˆ« »

```
#include <iostream>

int main() {
    // ((7 - 7) - 7) = -7
    std::cout << 7 - 7 - 7;

    // ((24 / 4) / 2) = 3
    std::cout << 24 / 4 / 2;
    return 0;
}
```

Increment / decrement



increment decrement
++a - - a a++ a - -

Increment / decrement

Prefix

1. increase the value
2. return the value

```
#include <iostream>

int main() {
    int x = 10;
    // 11
    std::cout << ++x;
    // 11
    std::cout << x;
}
```

Increment / decrement

Postfix

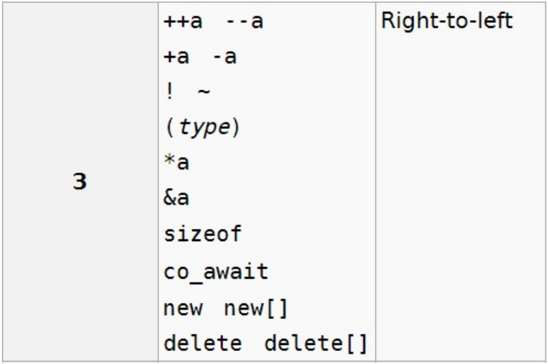
1. copy the value
2. increase the value
3. return the copy

```cpp
#include <iostream>

int main() {
    int x = 10;
    // 10
    std::cout << x++;
    // 11
    std::cout << x;
}
```

Let’s consider the combination of ”- -” and ”-”

```
#include <iostream>

int main() {
    int x = 10;
    std::cout << ---x;
    return 0;
}
```

We can see that - - and - have precedence 3 and both right-associative



3	`++a --a`
	`+a -a`
	`! ~`
	`(`*`type`*`)`
	`*a`
	`&a`
	`sizeof`
	`co_await`
	`new new[]`
	`delete delete[]`
		Right-to-left

Consequently the order of placing brackets is the following

```
#include <iostream>

int main() {
    int x = 10;
    std::cout << --(-x);
    return 0;
}
```

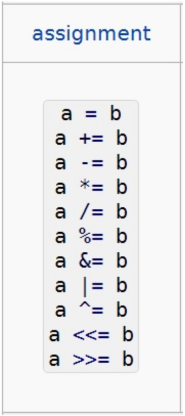
Yes, it raises an error, but the error is the same as it was at the beginning. If we have placed brackets in this way ”-(- -x)” than there would be no error. These two facts prove that the brackets were placed correctly.

Let’s consider another example

```cpp
#include <iostream>

int main() {
    int x = 10;
    // everything's fine here
    // output: 9
    std::cout << -(--x);
    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int x = 10;
    // there is no error, thus
    // brackets placing is
    // the following -(x--)
    std::cout << -x--;
    return 0;
}
```

Assignment



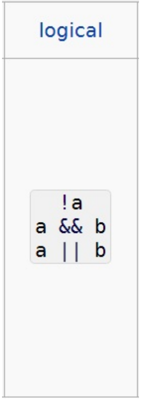
assignment
a = b a += b a -= b a *= b a /= b a %= b a &= b a |= b a ^= b a <<= b a >>= b

Assignment

```cpp
#include <iostream>

int main() {
    int x = 10;
    int y = 6;

    x &= y;
    // 2
    std::cout << x;
    return 0;
}
```

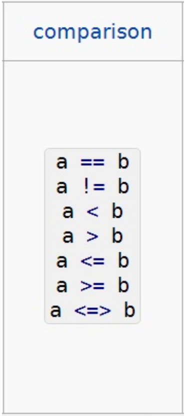
```cpp
#include <iostream>

int main() {
    int x = 10;
    int y = 6;

    x = x & y;
    // 2
    std::cout << x;
    return 0;
}
```

Logical



logical
!a a && b a || b

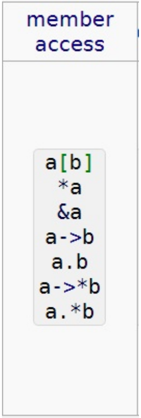
Comparison



comparison

```
a == b
a != b
a < b
a > b
a <= b
a >= b
a <=> b
```

Access



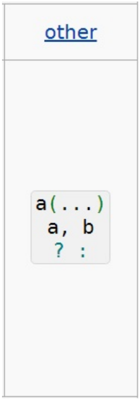
member access
a[b] *a &a a->b a.b a->*b a.*b

Access

```cpp
#include <iostream>

struct A {
    int x = 10;
};

int main() {
    A* p = new A;
    std::cout << p->x;
    std::cout << (*p).x;

    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int* p = new (5);
    // 5
    std::cout << *p;
    int x = 10;
    p = &x;
    // 10
    std::cout << p;

    return 0;
}
```

Other



other
a(...) a, b ? :

Ternary conditional

```cpp
#include <iostream>

int main() {
    int x = 0;
    int y = 1;

    std::cout << a > b ? a : b;
    return 0;
}
```

Comma

```
#include <iostream>

int main() {
    int n = 1;
    int m = (++n, std::cout << "n = " << n << '\n', ++n, 2*n);
    std::cout << "m = " << (++m, m) << '\n';

    return 0;
}
```

sizeof, alignof

- sizeof - yields the size in bytes for the given type
- alignof - yields the alignment in bytes

```
#include <iostream>

struct C {
    char x;
    int y;
};
int main() {
    // 1
    std::cout << alignof(char);
    // 4
    std::cout << alignof(int);
    // an alignment by int
    std::cout << sizeof(C);
    return 0;
}
```

lvalue and rvalue

Definition

- **lvalue** is an **expression** such that we can assign a value
- rvalue is an **expression** that is not lvalue expression

Note

Is is a naive definition. More accurately in the 2nd part of the course.

Function overloading

Definition

It is a definition of at least two functions in the same scope with the same name, different parameter lists and different cv-qualifiers.

Function overloading

Allowable

```
#include <iostream>

void func(double a) {}
void func(int a) {}

int main() {
    return 0;
}
```

Not allowable

```
#include <iostream>

void func(int a) {}
int func(int a) {}

int main() {
    return 0;
}
```

Default arguments

Only the trailing arguments can have default values

Allowable

```
#include <iostream>

void func(int a, int b = 0) {}

int main() {
    return 0;
}
```

Not allowable

```
#include <iostream>

void func(int b = 0, int a) {}

int main() {
    return 0;
}
```

Explicit type conversion

Problem

On the left is a pointer to double, which is assigned a the address of the float value.
We read 8 bytes (ptr to double), but the value is stored in 4 bytes.

C-cast

```
#include <iostream>

int main() {
    float x = 3.1;
    // UB
    double* y = (double*) &x;
    return 0;
}
```

static_cast

```
#include <iostream>

int main() {
    float x = 3.1;
    // CE
    double* y = static_cast<double*>(&x);
    return 0;
}
```

Summary

Use static_cast instead of C-cast. static_cast checks type compatibility.

Control flow statements

if

C++

```cpp
int main() {
    int x = 10;
    if (x) {
        int y = 20;
    }
    int z = 10;
    return 0;
}
```

x86-64 clang 10.0.0

```asm
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
    // compares the "x" with "0" and
    // stores result to ZF
    cmp     dword ptr [rbp - 8], 0
    // jump to label if ZF = 1
    je      .LBB0_2
    mov     dword ptr [rbp - 12], 20
.LBB0_2:
    xor     eax, eax
    mov     dword ptr [rbp - 16], 10
    pop     rbp
    ret
```

if, else

C++

```cpp
int main() {
    int x = 10;
    if (x) {
        int y = 20;
    } else {
        int y = 5;
    }
    int z = 10;
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
    cmp     dword ptr [rbp - 8], 0
    // conditional jump
    // associated with "if"
    je      .LBB0_2
    mov     dword ptr [rbp - 12], 20
    // jump associated with "else"
    jmp     .LBB0_3
.LBB0_2:
    mov     dword ptr [rbp - 16], 5
.LBB0_3:
    xor     eax, eax
    mov     dword ptr [rbp - 20], 10
    pop     rbp
```

Dangling else

```cpp
#include <iostream>

int main() {
    int x = 0;
    if (1)
        if (1)
            x = 1;
    else
        x = 2;
    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int x = 0;
    if (0)
        if (0)
            x = 1;
    else
        x = 2;
    return 0;
}
```

```cpp
#include <iostream>

int main() {
    int x = 0;
    if (1)
        if (0)
            x = 1;
    else
        x = 2;
    return 0;
}
```

Summary

Use braces and write explicitly

while

C++

```cpp
int main() {
    int x = 0;
    while(x < 1) {x++;}
    return 0;
}
```

x86-64 clang 10.0.0

```asm
main:
        push    rbp
        mov     rbp, rsp
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], 0
.LBB0_1:
        cmp     dword ptr [rbp - 8], 1
        jge     .LBB0_3
        mov     eax, dword ptr [rbp - 8]
        add     eax, 1
        mov     dword ptr [rbp - 8], eax
        jmp     .LBB0_1
.LBB0_3:
        xor     eax, eax
        pop     rbp
        ret
```

do-while

C++

```
int main() {
    int j = 0;
    do {
        j++;
    } while (j < 2);
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
        push    rbp
        mov     rbp, rsp
        mov     dword ptr [rbp - 4], 0
        mov     dword ptr [rbp - 8], 0
.LBB0_1:
        mov     eax, dword ptr [rbp - 8]
        add     eax, 1
        mov     dword ptr [rbp - 8], eax
        cmp     dword ptr [rbp - 8], 2
        jl      .LBB0_1
        xor     eax, eax
        pop     rbp
        ret
```

for

C++

```cpp
int main() {
    for(size_t x = 0; x < 1; x++)
    return 0;
}
```

x86-64 clang 10.0.0

```asm
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
.LBB0_1:
    cmp     dword ptr [rbp - 8], 1
    jge     .LBB0_4
    jmp     .LBB0_3
.LBB0_3:
    mov     eax, dword ptr [rbp - 8]
    add     eax, 1
    mov     dword ptr [rbp - 8], eax
    jmp     .LBB0_1
.LBB0_4:
    xor     eax, eax
    pop     rbp
    ret
```

switch

C++

```cpp
int main() {
    int x = 0;
    switch(x) {
        case 0 : {
            int y = 1;
            break;
        }
        default: {
            int y = 2;
        }
    }
    return 0;
}
```

x86-64 clang 10.0.0

```asm
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
    mov     eax, dword ptr [rbp - 8]
    test    eax, eax
    jne     .LBB0_2
    jmp     .LBB0_1
.LBB0_1:
    mov     dword ptr [rbp - 12], 1
    // jump associated with "break"
    jmp     .LBB0_3
.LBB0_2:
    mov     dword ptr [rbp - 16], 2
.LBB0_3:
    xor     eax, eax
    pop     rbp
```

switch

C++

```
int main() {
    int x = 0;
    switch(x) {
        case 0 : {
            int y = 1;
        }
        default: {
            int y = 2;
        }
    }
    return 0;
}
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 0
    mov     eax, dword ptr [rbp - 8]
    test    eax, eax
    jne     .LBB0_2
    jmp     .LBB0_1
.LBB0_1:
    mov     dword ptr [rbp - 12], 1
    // a piece of assembly code
    // associated with "default"
.LBB0_2:
    mov     dword ptr [rbp - 16], 2
    xor     eax, eax
    pop     rbp
    ret
```

break

break

Jump to label of the end of loop

C++

```cpp
int main() {
    int x = 10;
    while( x < 10) {
        break;
    }
    return 0;
}
```

x86-64 clang 10.0.0

```asm
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
    cmp     dword ptr [rbp - 8], 10
    jge     .LBB0_3
    // jump associated with "break"
    jmp     .LBB0_3
.LBB0_3:
    xor     eax, eax
    pop     rbp
    ret
```

continue

continue

Jump to label of the beginning of the loop

C++

```cpp
int main() {
    int x = 10;
    while( x < 10) {
        continue;
    }
    return 0;
}
```

x86-64 clang 10.0.0

```asm
main:
    push    rbp
    mov     rbp, rsp
    mov     dword ptr [rbp - 4], 0
    mov     dword ptr [rbp - 8], 10
.LBB0_1:
    cmp     dword ptr [rbp - 8], 10
    jge     .LBB0_3
    // jump associated with "continue"
    jmp     .LBB0_1
.LBB0_3:
    xor     eax, eax
    pop     rbp
    ret
```

Return

C++

```
    int main() {
        return 0;
    }
```

x86-64 clang 10.0.0

```
main:
    push    rbp
    mov     rbp, rsp
    xor     eax, eax
    mov     dword ptr [rbp - 4], 0
    // return 0
    pop     rbp
    ret
```