Accelerated entry into C++14

Romantic memories of awesome architecture (hand coded , relying on discipline, make your own tools)

Real world old school style results

#bugfest

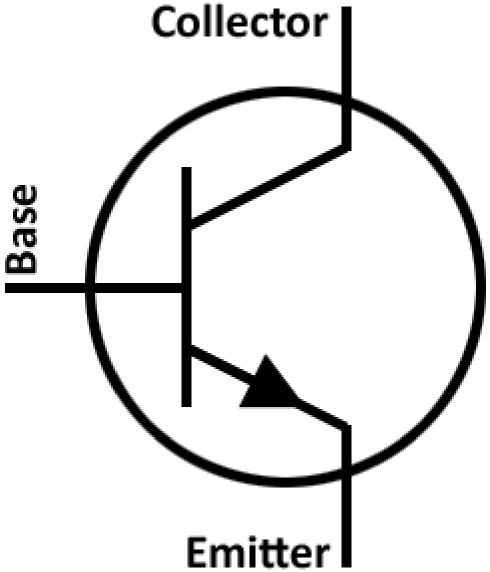
Why do construction workers succeed at anything?

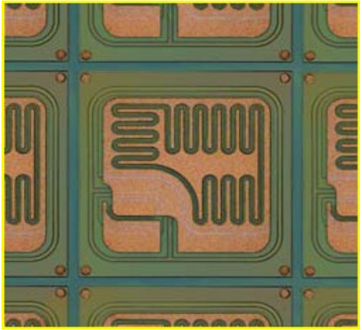
more awesome pictures at THEMETAPICTURE.COM

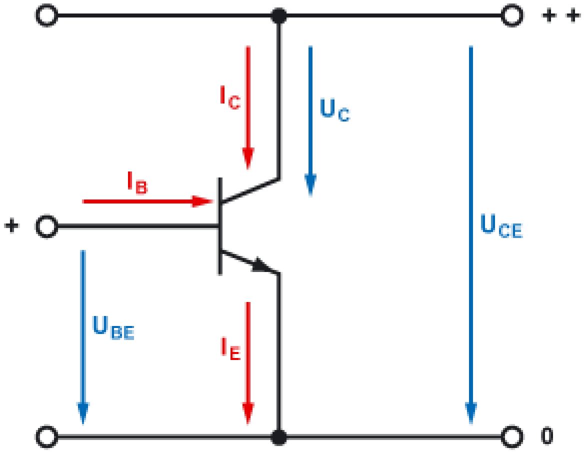
Encapsulation of expertise

Sheldon like precise description in the flash like speed

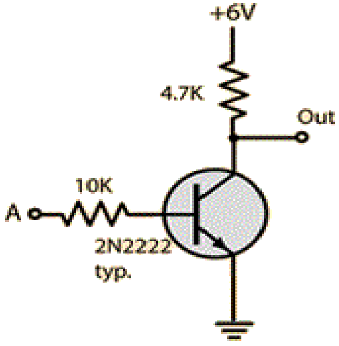
Bare metal low level

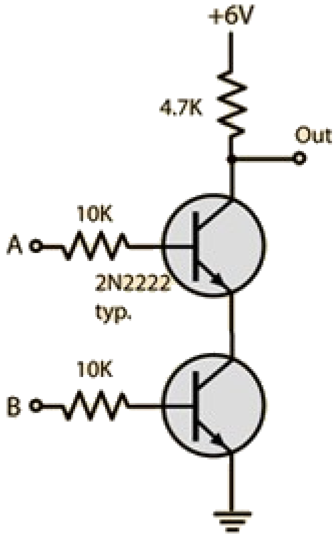


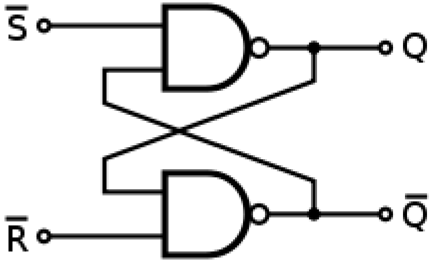




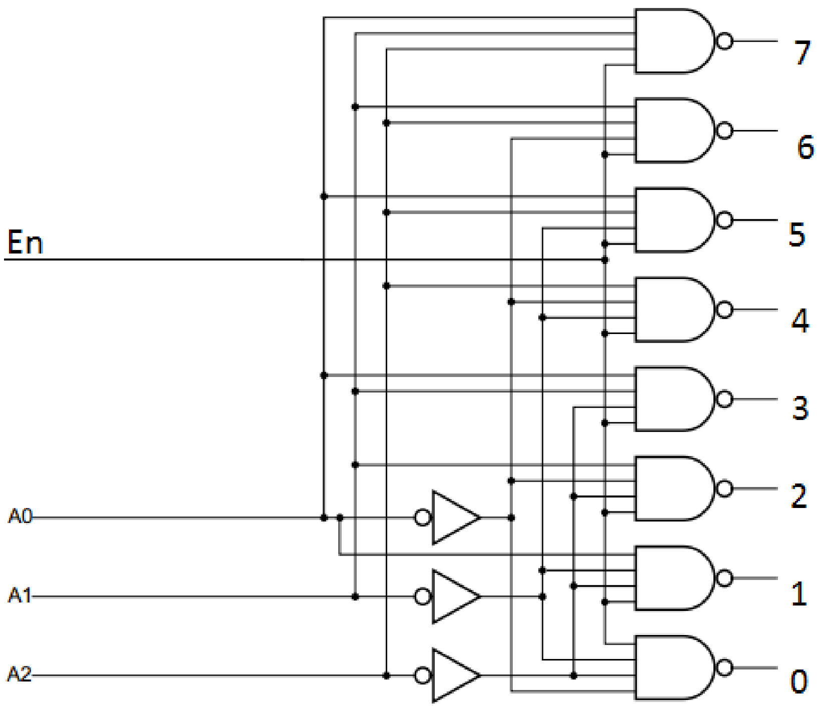
Gates, Flipflops



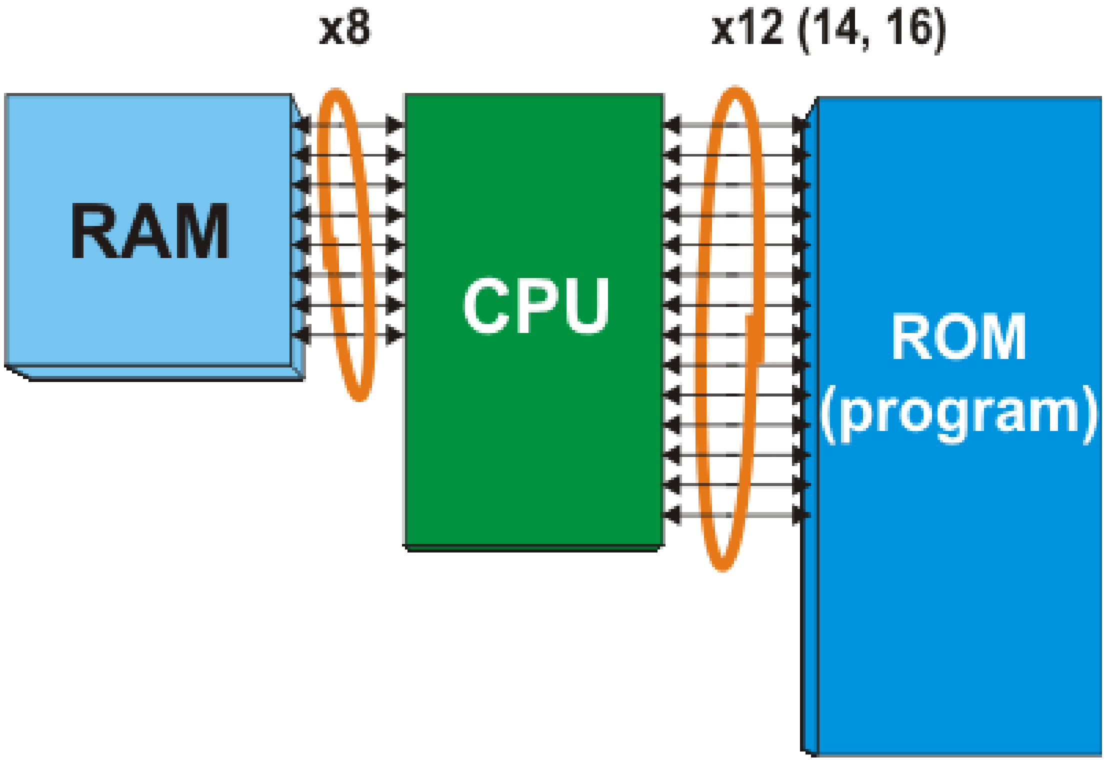




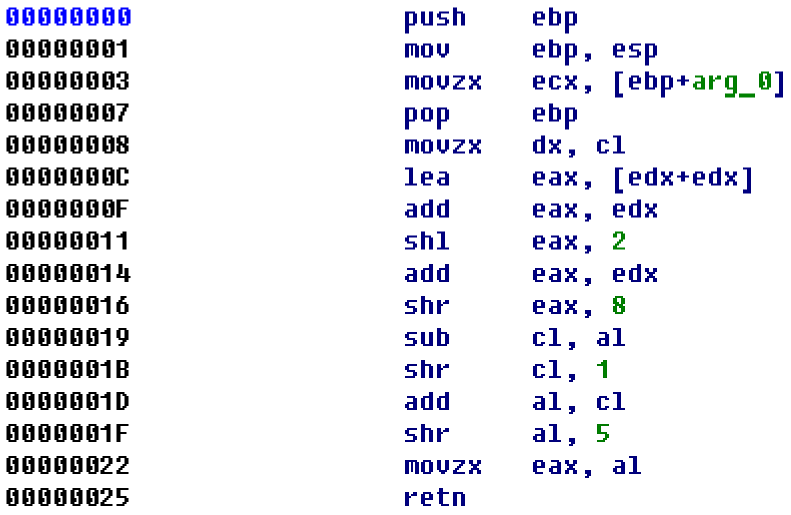
Addressing, RAM



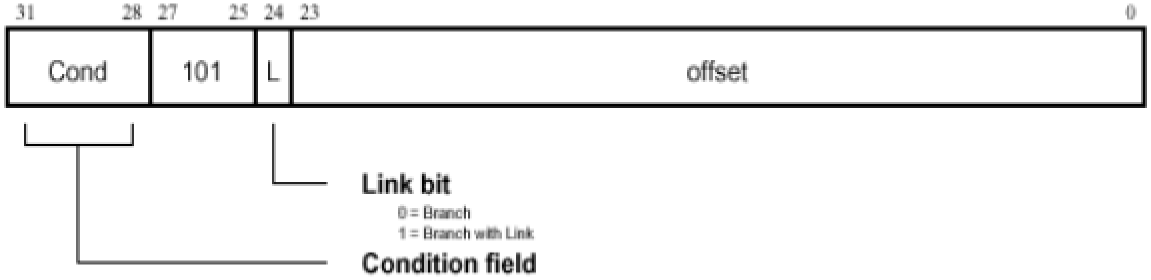
Processor



Assembler



```
00000000                push    ebp
00000001                mov     ebp, esp
00000003                movzx   ecx, [ebp+arg_0]
00000007                pop     ebp
00000008                movzx   dx, cl
0000000C                lea     eax, [edx+edx]
0000000F                add     eax, edx
00000011                shl     eax, 2
00000014                add     eax, edx
00000016                shr     eax, 8
00000019                sub     cl, al
0000001B                shr     cl, 1
0000001D                add     al, cl
0000001F                shr     al, 5
00000022                movzx   eax, al
00000025                retn
```

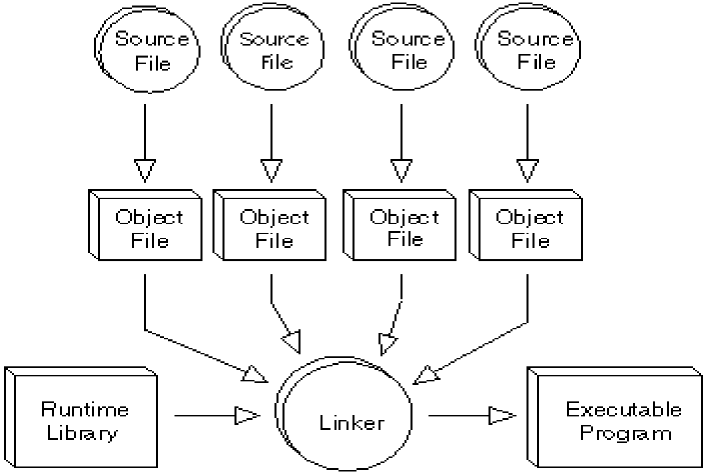


Assembler 2

- simple commands like add, branch, xor, move etc.
- different commands for each processor
- even a simple program can be millions of lines of code!
- memory access done by hand!

Compiler

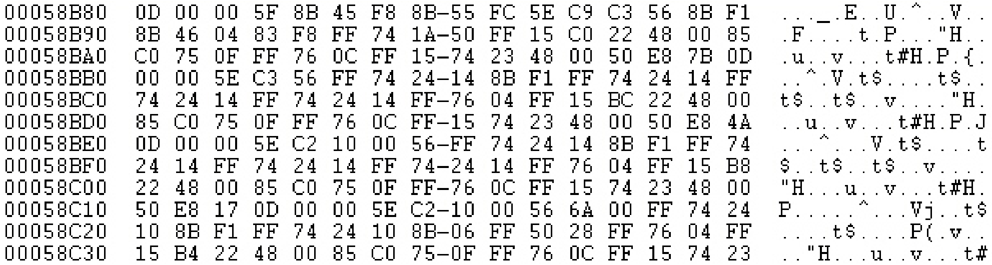
- translates code from a higher level programming language into assembler
- keeps track of memory automatically*
- can translate the same code into assembler for different processors



C the original sin of C++

- C is a subset of C++
- There is nothing C can do which C++ cannot
- C is missing mechanisms used for safety checks
- C is missing mechanisms for generic code
- C is missing mechanisms for zero cost abstraction
- You can write bad code in both C and C++ but its often impossible to write good, safe library code in C
- There is a myth that C is more efficient which is propagated exclusively by those who cannot program well in C++

Types vs. Values



```
00058B80  0D 00 00 5F 8B 45 F8 8B-55 FC 5E C9 C3 56 8B F1  ..._.E..U.^..V..
00058B90  8B 46 04 83 F8 FF 74 1A-50 FF 15 C0 22 48 00 85  .F....t.P..."H..
00058BA0  C0 75 0F FF 76 0C FF 15-74 23 48 00 50 E8 7B 0D  .u..v...t#H.P.{.
00058BB0  00 00 5E C3 56 FF 74 24-14 8B F1 FF 74 24 14 FF  ..^.V.t$....t$..
00058BC0  74 24 14 FF 74 24 14 FF-76 04 FF 15 BC 22 48 00  t$..t$..v...."H.
00058BD0  85 C0 75 0F FF 76 0C FF-15 74 23 48 00 50 E8 4A  ..u..v...t#H.P.J
00058BE0  0D 00 00 5E C2 10 00 56-FF 74 24 14 8B F1 FF 74  ...^...V.t$....t
00058BF0  24 14 FF 74 24 14 FF 74-24 14 FF 76 04 FF 15 B8  $..t$..t$..v....
00058C00  22 48 00 85 C0 75 0F FF-76 0C FF 15 74 23 48 00  "H...u..v...t#H.
00058C10  50 E8 17 0D 00 00 5E C2-10 00 56 6A 00 FF 74 24  P.....^...Vj..t$
00058C20  10 8B F1 FF 74 24 10 8B-06 FF 50 28 FF 76 04 FF  ....t$....P(.v..
00058C30  15 B4 22 48 00 85 C0 75-0F FF 76 0C FF 15 74 23  .."H...u..v...t#
```

Types:

- contain information about the interpretation of data
- type information is used by the compiler in order to generate the correct code
- type information often decides which overload (version) of a function is used when manipulating data
- contains the size (in bytes) of the data

Data is just raw bits

Low level data types in C++:

bool: used to represent a true/false value. Actually uses more than one bit.
int / long: used to represent whole numbers.
float / double: used to represent floating point numbers.
enum class: used to represent lists of things which are not numbers (but are encoded as numbers)
char: used to represent letters and typographical symbols
string: used to represent strings of letters (words, sentences etc.)

Real code:

```
int i{ 4 };
i += 7;
double d{ 100.42 / i };
bool b{ d > i };

enum class CatName { Garfield, Fuzzy, KickMe };
enum class KidName { Bob, Joe, KickMe };
CatName cat{ CatName::KickMe };
KidName kid{ KidName::KickMe };
cat = i; //error
cat = kid; //error

char c{ 'a' };
std::string s{ "foo" };
char firstLetter = s[0];
```

Arrays

```
KidName redneckFamily[10]{
    KidName::Bob,
    KidName::Bob,
    KidName::Joe,
    KidName::Bob,
    KidName::KickMe,
    KidName::KickMe,
    KidName::Bob,
    KidName::Bob,
    KidName::Joe,
    KidName::Bob };
auto youngest{ redneckFamily[10] };
```

The two hardest things in programming are off by one errors.

Keywords

alignas (since C++11)
alignof (since C++11)
and
and_eq
asm
auto(1)
bitand
bitor
bool
break
case
catch
char
char16_t (since C++11)
char32_t (since C++11)
class
compl
concept (concepts TS)
const
constexpr (since C++11)
const_cast
continue
decltype (since C++11)
default(1)
delete(1)
do
double
dynamic_cast
else
enum
explicit
export(1)
extern
false
float
for
friend
goto
if
inline
int
long
mutable
namespace
new
noexcept (since C++11)
not
not_eq
nullptr (since C++11)
operator
or
or_eq
private
protected
public
register
reinterpret_cast
requires (concepts TS)
return
short
signed
sizeof
static
static_assert (since C++11)
static_cast
struct
switch
template
this
thread_local (since C++11)
throw
true
try
typedef
typeid
typename
union
unsigned
using(1)
virtual
void
volatile
wchar_t
while
xor
xor_eq

Functions

```cpp
int square(int in) {
    return in*in;
}

auto s{ square(4) };

KidName operator *(KidName lhs, KidName rhs) {
    return KidName::KickMe;
}

KidName k{ KidName::Joe };
auto kk = k*k;

bool shouldIKickIt(CatName cat) {
    return true;
}

bool shouldIKickIt(KidName kid) {
    return kid == KidName::KickMe;
}
```

Arithmetic operators

Operator name		Syntax	Can overload	Included in C	Prototype examples	
					As member of K	Outside class definitions
Basic assignment		`a = b`	Yes	Yes	`R& K::operator =(S b);`	N/A
Addition		`a + b`	Yes	Yes	`R K::operator +(S b);`	`R operator +(K a, S b);`
Subtraction		`a - b`	Yes	Yes	`R K::operator -(S b);`	`R operator -(K a, S b);`
Unary plus (integer promotion)		`+a`	Yes	Yes	`R K::operator +();`	`R operator +(K a);`
Unary minus (additive inverse)		`-a`	Yes	Yes	`R K::operator -();`	`R operator -(K a);`
Multiplication		`a * b`	Yes	Yes	`R K::operator *(S b);`	`R operator *(K a, S b);`
Division		`a / b`	Yes	Yes	`R K::operator /(S b);`	`R operator /(K a, S b);`
Modulo (integer remainder)[a]		`a % b`	Yes	Yes	`R K::operator %(S b);`	`R operator %(K a, S b);`
Increment	Prefix	`++a`	Yes	Yes	`R& K::operator ++();`	`R& operator ++(K& a);`
	Postfix	`a++`	Yes	Yes	`R K::operator ++(int);` Note: C++ uses the unnamed dummy-parameter `int` to differentiate between prefix and suffix increment operators.	`R operator ++(K& a, int);`
Decrement	Prefix	`--a`	Yes	Yes	`R& K::operator --();`	`R& operator --(K& a);`
	Postfix	`a--`	Yes	Yes	`R K::operator --(int);` Note: C++ uses the unnamed dummy-parameter `int` to differentiate between prefix and suffix decrement operators.	`R operator --(K& a, int);`

Comparison operators/relational operators

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Equal to	a **==** b	Yes	Yes	bool K::operator ==(S const& b);	bool operator ==(K const& a, S const& b);

Not equal to	a **!=** b
	a **not_eq** b[b]
		Yes	Yes	bool K::operator !=(S const& b);
				bool K::operator !=(S const& b) const;
					bool operator !=(K const& a, S const& b);

Greater than	a **>** b	Yes	Yes	bool K::operator >(S const& b) const;	bool operator >(K const& a, S const& b);
Less than	a **<** b	Yes	Yes	bool K::operator <(S const& b)const;	bool operator <(K const& a, S const& b);
Greater than or equal to	a **>=** b	Yes	Yes	bool K::operator >=(S const& b) const;	bool operator >=(K const& a, S const& b);
Less than or equal to	a **<=** b	Yes	Yes	bool K::operator <=(S const& b);	bool operator <=(K const& a, S const& b);

Logical operators

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Logical negation (NOT)	`!a` `not a`[b]	Yes	Yes	`R K::operator !();`	`R operator !(K a);`
Logical AND	`a && b` `a and b`[b]	Yes	Yes	`R K::operator &&(S b);`	`R operator &&(K a, S b);`
Logical OR	`a || b` `a or b`[b]	Yes	Yes	`R K::operator ||(S b);`	`R operator ||(K a, S b);`

Bitwise operators

Operator name	Syntax	Can overload	Included in C	Prototype examples	
				As member of K	Outside class definitions
Bitwise NOT	`~a` `compl a`[b]	Yes	Yes	`R K::operator ~();`	`R operator ~(K a);`
Bitwise AND	`a & b` `a bitand b`[b]	Yes	Yes	`R K::operator &(S b);`	`R operator &(K a, S b);`
Bitwise OR	`a | b` `a bitor b`[b]	Yes	Yes	`R K::operator |(S b);`	`R operator |(K a, S b);`
Bitwise XOR	`a ^ b` `a xor b`[b]	Yes	Yes	`R K::operator ^(S b);`	`R operator ^(K a, S b);`
Bitwise left shift[c]	`a << b`	Yes	Yes	`R K::operator <<(S b);`	`R operator <<(K a, S b);`
Bitwise right shift[c][d]	`a >> b`	Yes	Yes	`R K::operator >>(S b);`	`R operator >>(K a, S b);`

Compound assignment operators

Operator name	Syntax	Meaning	Can overload	Included in C	Prototype examples	
					As member of K	Outside class definitions
Addition assignment	a **+=** b	a **=** a **+** b	Yes	Yes	R& K::operator +=(S b);	R& operator +=(K a, S b);
Subtraction assignment	a **-=** b	a **=** a **-** b	Yes	Yes	R& K::operator -=(S b);	R& operator -=(K a, S b);
Multiplication assignment	a ***=** b	a **=** a ***** b	Yes	Yes	R& K::operator *=(S b);	R& operator *=(K a, S b);
Division assignment	a **/=** b	a **=** a **/** b	Yes	Yes	R& K::operator /=(S b);	R& operator /=(K a, S b);
Modulo assignment	a **%=** b	a **=** a **%** b	Yes	Yes	R& K::operator %=(S b);	R& operator %=(K a, S b);
Bitwise AND assignment	a **&=** b a **and_eq** b[b]	a **=** a **&** b	Yes	Yes	R& K::operator &=(S b);	R& operator &=(K a, S b);
Bitwise OR assignment	a **|=** b a **or_eq** b[b]	a **=** a **|** b	Yes	Yes	R& K::operator |=(S b);	R& operator |=(K a, S b);
Bitwise XOR assignment	a **^=** b a **xor_eq** b[b]	a **=** a **^** b	Yes	Yes	R& K::operator ^=(S b);	R& operator ^=(K a, S b);
Bitwise left shift assignment	a **<<=** b	a **=** a **<<** b	Yes	Yes	R& K::operator <<=(S b);	R& operator <<=(K a, S b);
Bitwise right shift assignment[d]	a **>>=** b	a **=** a **>>** b	Yes	Yes	R& K::operator >>=(S b);	R& operator >>=(K a, S b);

In the beginning there was void, then god created main

```
int main(int argc, const char** argv)
{
    if (argc == 0) {
        return 1.4;
    }
    else if (argc == 1) {
        return false;
    }
    else {
        return square(argc);
    }
}
```

Data Structures

```
struct MyRedneckHouse {
    KidName kid_;
    CatName cat_;
    int taxDebt_;
    int bankDebt_;
    int liquerBottlesInFrontLawn_;
    int junkCarsInFrontLawn_;
    bool americanFlag_;
};

struct SouthernRedneckHouse : MyRedneckHouse {
    bool confediratеFlag_;
};
```

Member functions

```
struct MyRedneckHouse {
    KidName kid_;
    CatName cat_;
    int taxDebt_;
    int bankDebt_;
    int liquerBottlesInFrontLawn_;
    int junkCarsInFrontLawn_;
    bool americanFlag_;
    void buyFlag() {
        bankDebt_ += 10;
        americanFlag_ = true;
    }
};

    MyRedneckHouse house;
    house.buyFlag();

    MyRedneckHouse house2{};
    house2.buyFlag();
```

What most books start with

```
int main(int argc, const char* argv[])
{
     std::string s{ "Hello world" };
     std::cout << s;
}

int main(int argc, const char** argv)
{
     std::cout << "Enter your name:" << std::endl;
     std::string s{};
     std::cin >> s;
     std::cout << "hello " << s << std::endl;
}
```

Homework

- Make github.com account
- go to github.com/porkybrain/accelerated-entry-into-cpp14
- watch this repository
- check out useful links in this repository
- write a program which takes a users height and weight and shows their BMI