Building Blocks

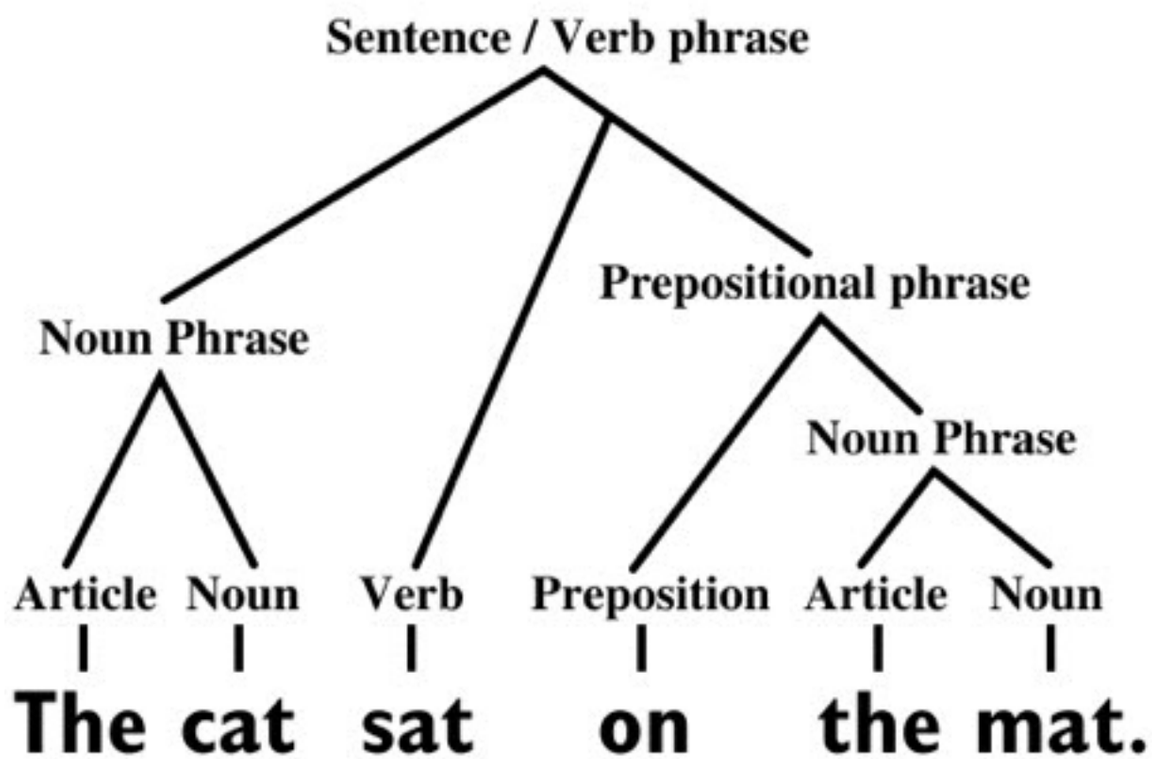
"to get a deeper understanding of the language"

Deep C - a 3 day course

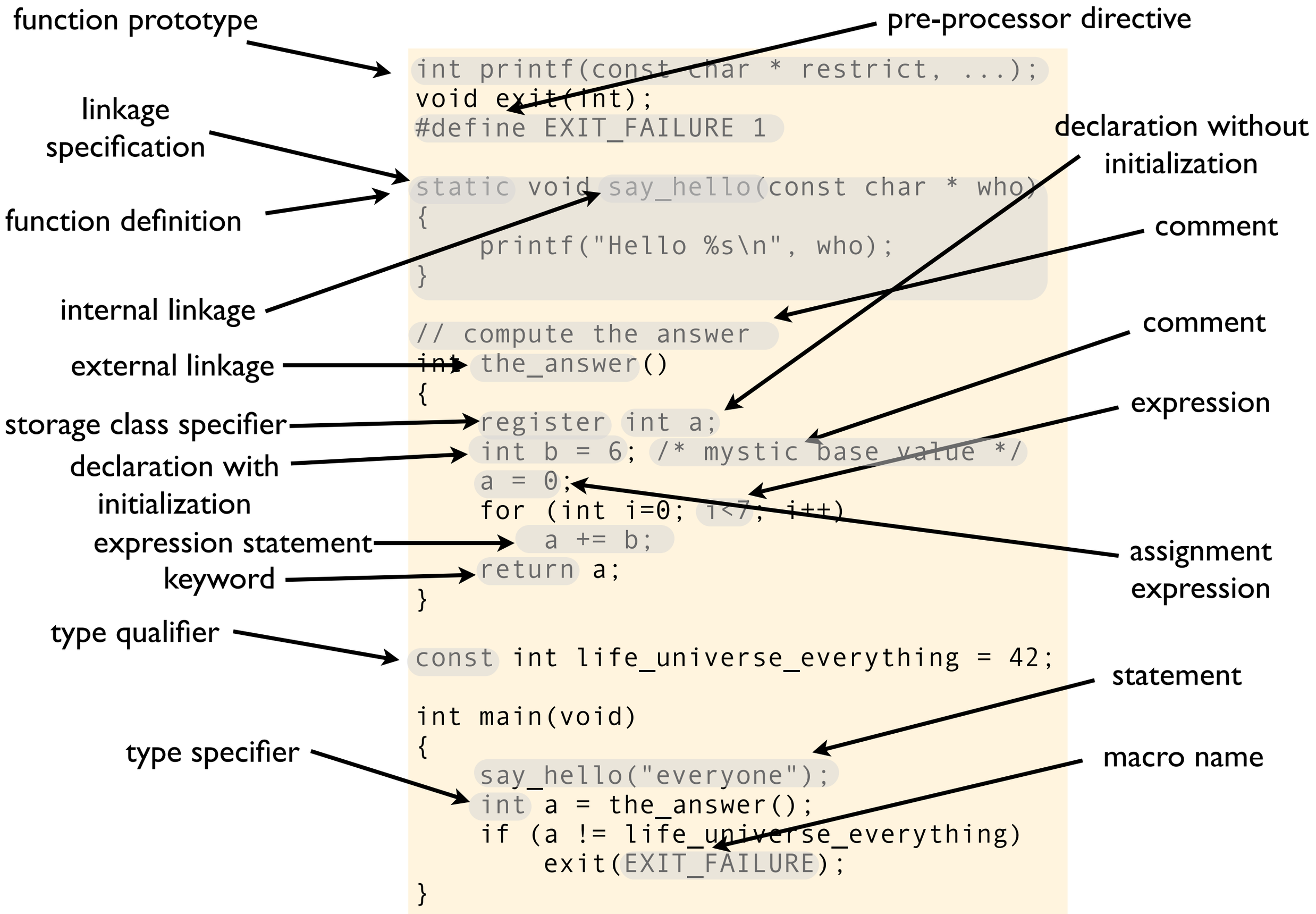
Jon Jagger & Olve Maudal

To get a deep understanding of any language you need to be able to ‘break’ it down and analyse it. You need to recognize the *words* used when experts are discussing among themselves.

Basic constituent structure analysis of a sentence:



http://en.wikipedia.org/wiki/File:Basic_constituent_structure_analysis_English_sentence.svg


function prototype
pre-processor directive
linkage specification
declaration without initialization
function definition
comment
internal linkage
comment
external linkage
expression
storage class specifier
declaration with initialization
expression statement
assignment expression
keyword
type qualifier
statement
type specifier
macro name
int printf(const char * restrict, ...);
void exit(int);
#define EXIT_FAILURE 1
static void say_hello(const char * who)
{
printf("Hello %s\n", who);
}
// compute the answer
int the_answer()
{
register int a;
int b = 6; /* mystic base value */
a = 0;
for (int i=0; i<7; i++)
a += b;
return a;
}
const int life_universe_everything = 42;
int main(void)
{
say_hello("everyone");
int a = the_answer();
if (a != life_universe_everything)
exit(EXIT_FAILURE);
}

```
if (answer < 0)
    answer = 42;
```

this is what you see

lexical analyser

```
if
(
answer
<
0
)
answer
=
42
;
```

this is what the compiler sees

```
if (answer < 0)
    answer = 42;
```

lexical analyser

`if`	keyword if
`(`	punctuator (
`answer`	identifier answer
`<`	operator <
`0`	constant 0
`)`	punctuator)
`answer`	identifier answer
`=`	operator =
`42`	constant 42
`;`	punctuator ;

There are 6 classes of tokens in a C program

token	examples
keyword	`if struct restrict` …
identifier	`the_answer main i`
constant	`9 6f 013 '9' 9.6`
string literal	"Hello"
operator	`+ - * / % = , () []` …
punctuator	`; = , { } ( ) []`

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

keywords

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

keywords

identifiers

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

keywords

constants

identifiers

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

keywords

constants

identifiers

string literals

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

keywords

constants

punctuators

identifiers

string literals

```c
#include <stdio.h>

static int calc(int a, int b, int c)
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

keywords	constants	punctuators
identifiers	string literals	operators

```c
#include <stdio.h>

static int calc(int a,
{
    return a * b / c;
}

int universe = 7;
static int life(void) { return 6; }
int everything(void) { return 1; }

int main(void)
{
    int a = calc(universe, life(), everything());
    printf("The answer is %d\n", a);
}
```

but what about this one?

this is a preprocessor directive, and is not really part of the language. We will deal with this later.

keywords

constants

punctuators

identifiers

string literals

operators

Keywords

auto	int
break	long
case	register
char	return
const	short
continue	signed
default	sizeof
do	static
double	struct
else	switch
enum	typedef
extern	union
float	unsigned
for	void
goto	volatile
if	while

C89

_Bool
_Complex
_Imaginary
inline
restrict

+C99

_Alignas
_Alignof
_Atomic
_Generic
_Noreturn
_Static_assert
_Thread_local

+C11

Identifiers

Rules
- made of letters, digits and underscores
- can't start with a digit
- case sensitive

```
variable1
802id
```

Recommendations
- don't start with underscore (reserved)
- use case consistently
- avoid abbrs
- use standard spelling
- don't use hungarian

```
_variable
vrbls
iVariable

CamelCase
snake_case
mixedCase
```

```
int year = 1992;
if (is_leap_year(year))
    do_extra_maintainance();
```

```
int year = 1992;
if (IsLeapYear(year))
    DoExtraMaintainance();
```

```
int Year = 1992;
if (IsLeapYear(Year))
    do_extra_maintainance();
```

Follow the local de facto standards - but when in doubt, follow K&R

Identifiers

larger scope ←→ longer identifier
smaller scope ←→ shorter identifier

```
for (size_t loop_index = 0; loop_index < 42; loop_index++)
{
    widgetize(wibbles[loop_index]);
}
```

better

```
for (size_t index = 0; index < 42; index++)
{
    widgetize(wibbles[index]);
}
```

better

```
for (size_t at = 0; at < 42; at++)
{
    widgetize(wibbles[at]);
}
```

Quick...What does the following code print?

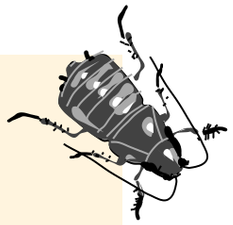
```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a =- 2;
    printf("%d\n", a);
}
```

Quick...What does the following code print?

```
#include <stdio.h>

int main(void)
{
    int a = 44;
    a =- 2;
    printf("%d\n", a);
}
```



lexical analyser

this is what the compiler sees

```
...
a
=
-
2
;
...
```

identifier a

operator =

operator -

constant 2

```
-2
```

but you do get a warning from the compiler?

```
$ gcc foo.c && ./a.out
-2
$ gcc -Wall -Wextra -pedantic foo.c && ./a.out
-2
```

Summary

- use accurate terminology
- what you see is often not what the compiler sees!
  - preprocessor
  - tokenization
- 6 kinds of token
  - keywords
  - identifiers
  - constants
  - string literals
  - punctuators
  - operators