C Basics

Computer Systems, No relevant Sections

The C Programming Language (K&R), Chapter 4, Appendix A

Function in C

```
int cent_to_far(int t) {
        return (t*9)/5+32;
}
```

http://www.pronk.com/samples/projects/021$Function_Machine/Function_Machine.HTML

General C syntax

- Comments ignored
  - Block comments start with "/*" and end with "*/" – don't nest!
  - Line comments start with "//" and end with end of line
- White space (blanks, tabs, newlines) is ignored
  - only delimits tokens
  - "int var;" is different from "intvar;", but is the same as "int var ;" and is also the same as "int

var

;"

C Statements

- A statement is a list of tokens that ends with a semi-colon
  - a=a+3;
  - int c=7;
  - int cent_to_far(int c);
- A C statement may span more than one line in the file
  - int MyLongFunctionNamedFunctionThatTakesLotsOfArguments(
    int arg1, int arg2, int arg3, int arg4,
    int arg5, int arg6, int arg7, int arg8);
- There may be more than one statement on one line in the file
  - int a=5; a=a+6; int b; b=cent_to_far(a);

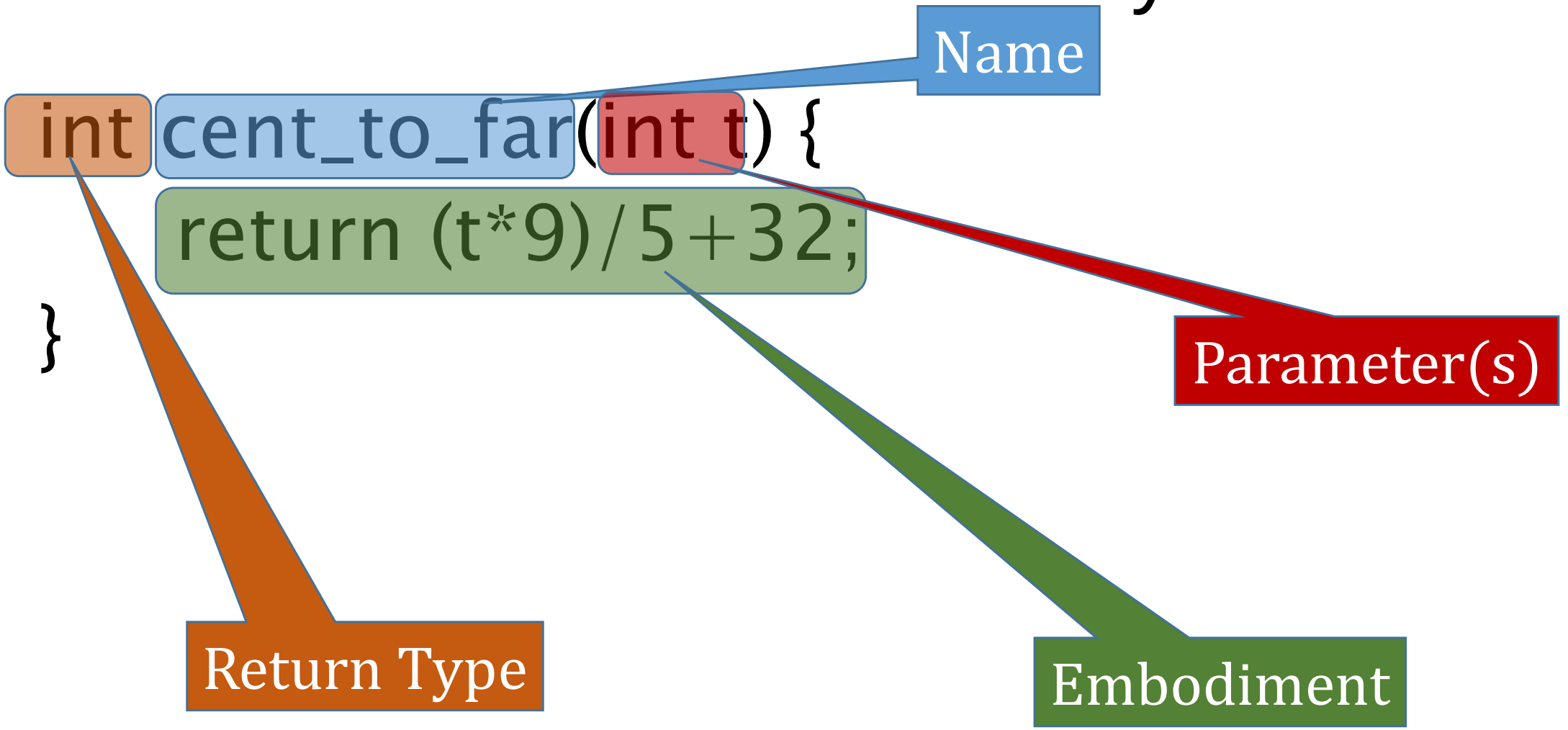
C Blocks

- A block of statements is a list of statements, surrounded by { and }
  - { - Left curly brace
  - } – Right curly brace
- A block can be used anywhere a statement can be used
- Blocks of statements can be nested
  - { statement1; { statement2; statement3; } statement4; }

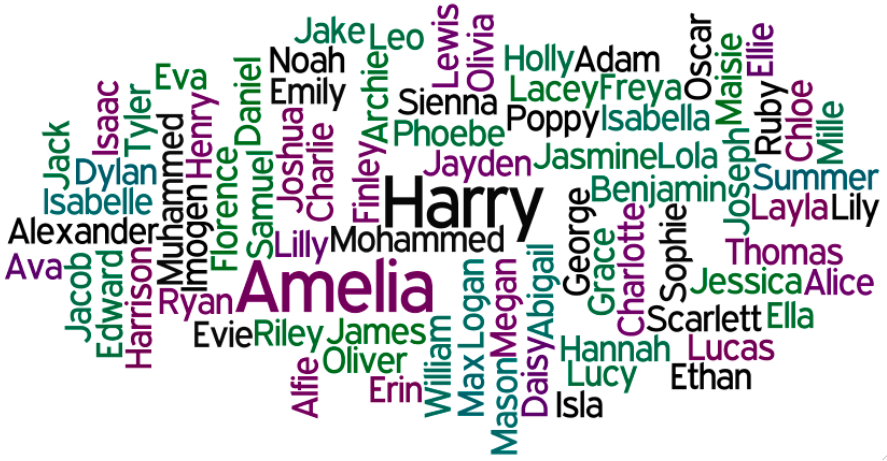
C File

- Boilerplate Pre-amble stuff
  - Pre-processor directives (start with “#”)
  - Global Variable declaration statements
  - Function declaration statements
- **Function definitions**

C Function Definition Anatomy



Identifiers



- Function, parameter, and variable names
- Must start with a letter or an underscore
  - Underscores usually avoided
- May not contain white space
- After the first letter, can be any number, letter, or underscore
- Identifiers are case sensitive
  - "polyArea" is different from "PolyArea"
- May not be a keyword
- Choose names that are descriptive, and easy to type
  - "Be4aTgh9_fr37200aBy" is probably not a good choice

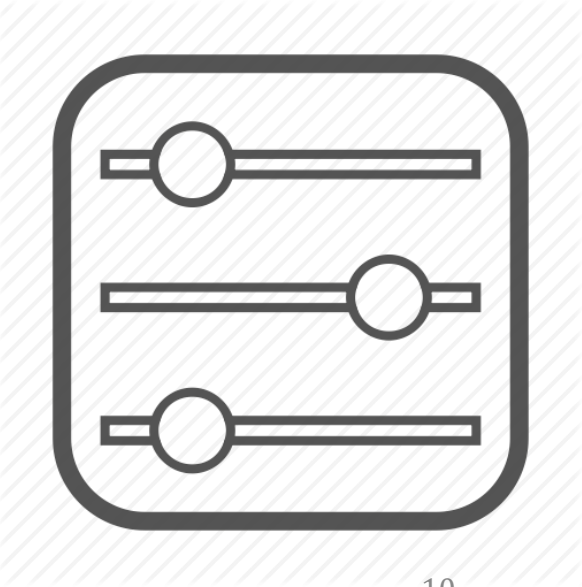
Keywords

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

Function Parameters

- Comma separated list of data passed in to the function
- Each entry in the list specifies the “type” and “name” of one parameter
- The value of the parameter can be referenced by its name in the function embodiment

- Terminology: Parameter from inside the function
- Argument when calling the function



Statements in a Function Embodiment

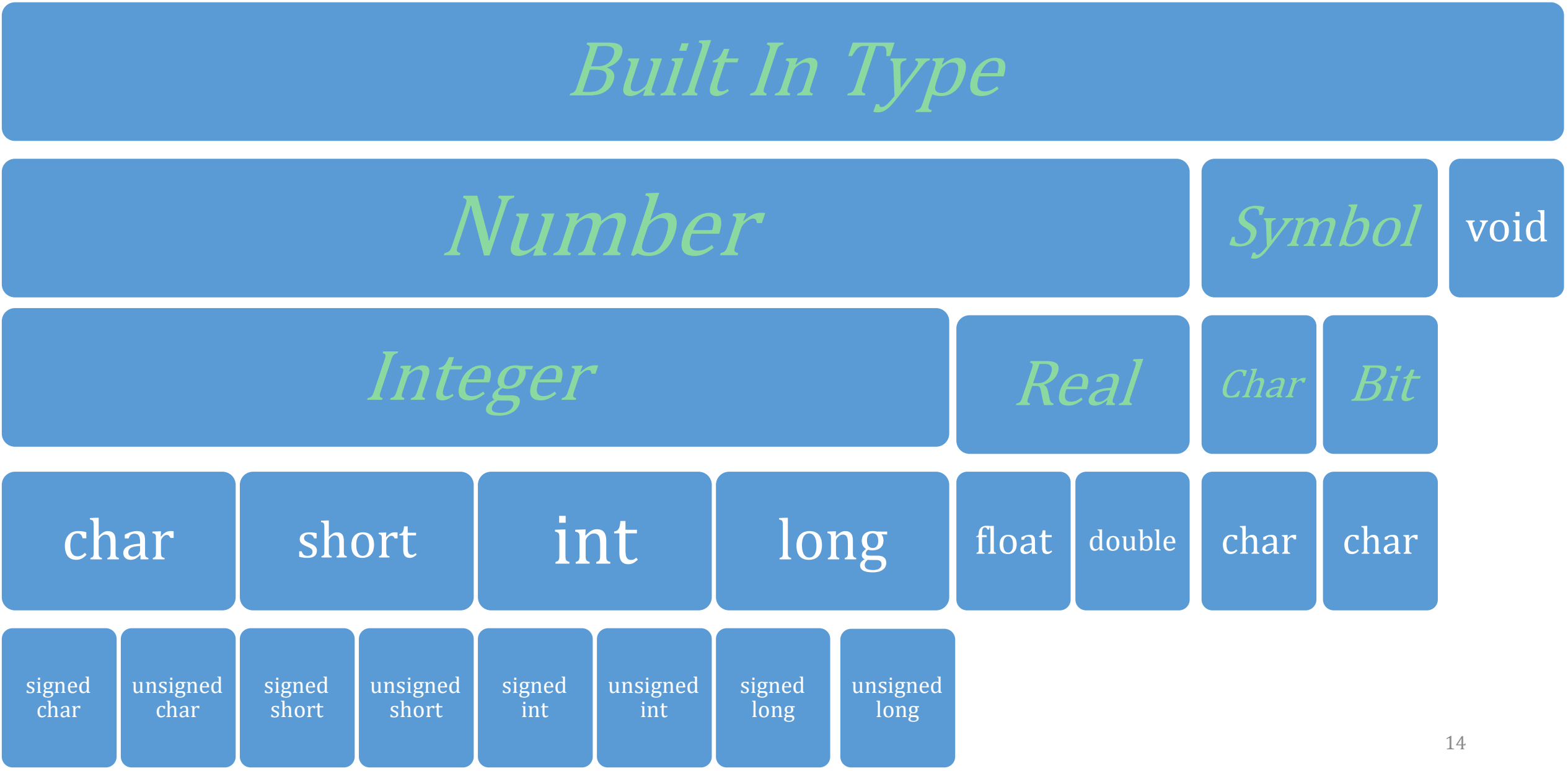
- Variable Declaration Statements
- Expressions statements <expression>;
  - Assignment statements: <variable>=<expression>;
- Control Statements: if/then/else, do while, for, return, ….

Variable Declaration Statements

- <type> <name> [=<initial value>];
- Required for each variable in your program
- Tells the compiler
  - The "type" of the variable, e.g. "int" or "float"
  - The name of the variable
  - Optionally, the initial value of the variable
- For example: `int x=3;`
- Basic Types: char, int, float

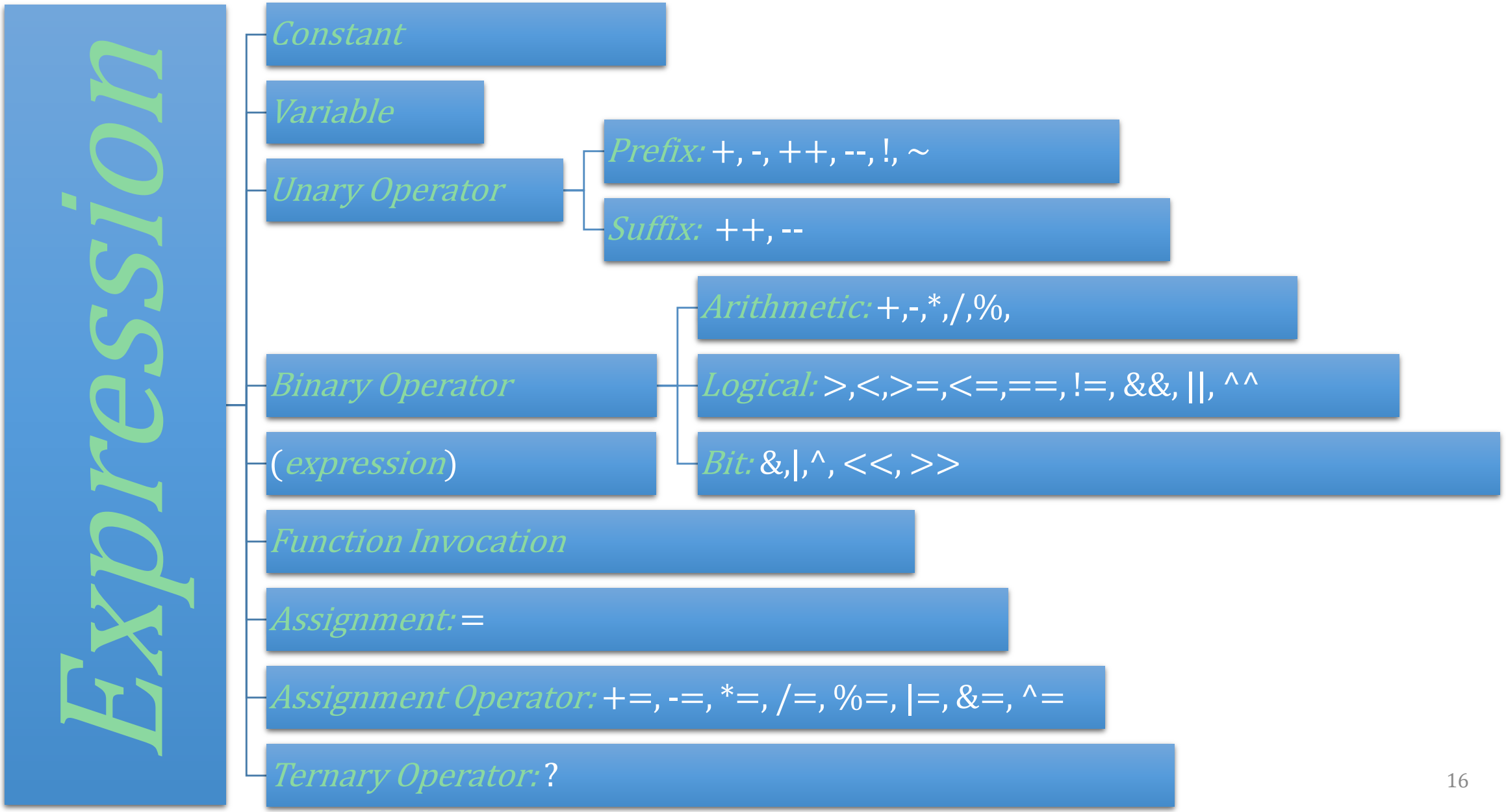
Variables in Functions

- Variable Declaration within function (usually at top)
- “Automatic” storage class by default
- New variable each time function starts
- Initialized when function starts (if initializer specified)
- No longer available when function ends


Built In Type
Number
Symbol
void
Integer
Real
Char
Bit
char
short
int
long
float
double
char
char
signed char
unsigned char
signed short
unsigned short
signed int
unsigned int
signed long
unsigned long

Example Expression Statements

```
int x; int y; /* Variable Declarations */
x=13; /* Constant assignment */
y=x; /* Variable assignment */
y=-x; /* Unary Operator */
y=x*3; /* Binary Operator */
y=(x*3)+7; /* Parenthesis */
y=atoi(argv[2]); /* Function invocation */
x=y=7; / Assignment operator */
x*=3; /* Compound Assignment Operator */
```


Expression
Constant
Variable
Unary Operator
Prefix: +, -, ++, --, !, ~
Suffix: ++, --
Binary Operator
Arithmetic: +,-,*,/,%,
Logical: >,<,>=,<=,==, !=, &&, ||, ^^
Bit: &,|,^, <<, >>
(expression)
Function Invocation
Assignment: =
Assignment Operator: +=, -=, *=, /=, %=, |=, &=, ^=
Ternary Operator: ?

C Operators with Side Effects

- y=++x; // Prefix increment
- x=x+1; // increment first
  y=x; // then assign
- y=x++; // Suffix Increment
- y=x; // assign
  x=x+1; // then increment

Assignment Operators

x <op>=y;

...is the same as...

x = x <op> y

- x+=3;
- x|=0x02;
- x/=y+3

x = x + 3;

x = x | 0x02;

x = x / (y+3) ;

Expression Statements / Equals Operator

- Assignment statements are really expression statements
- "Value" of an assignment is the value of the LHS
- x=y=z=0; // initialize x, y, and z to zero

- If a statement is just an expression, its value is discarded

```
printf("This is a test\n"); // discards 15… number of chars printed
a=3; // discards 3… the value of "a"
(3+21)/6; // discards 4… the value of (3+21)/6
```

Why not more operators?

- Standard Library Functions
  - stdlib.h
    - Conversion – atoi, atol, atof, strtol, strtoul, strtod
    - math – abs, labs, div, ldiv, rand
  - math.h
    - Trigonometry – sin, cos, tan, asin, acos, atan, sinh, cosh, tanh
    - Exponents – sqrt, pow, log, exp, log10, …
    - Rounding – fabs, floor, ciel
  - …

Operator Precedence

( )	[ ]	->	.							
!	-	++	--	+	-	*	&	(type)	sizeof	
*	/	%								
+	-									
<<	>>									
<	<=	>=	>							
==	!=									
&	^	|								
&&	||									
?										
=	+=	-=	*=	/=	%=	&=	^=	|=	<<=	>>=
,										

C function invocation

- Arguments must match parameter types
  - as specified in function signature
- For instance;

```
int y; y=area('r',3,4) * 2; // two 3x4 rectangles
```

- Function must be **declared** before it can be invoked
- One way of declaring a function is to define the function.
- This leads to upside down code.

Example of Upside Down Code

```
int square(int x) { return x*x; }
int poly(int x) {
        return square(x) + 2*x + 4;
}
int main() {
        int x=7;
        if (poly(x)<9) return 1;
        return 0;
}
```

C function declaration

Function prototype

- <result type> <function name>(<arglist>) ;
- Optional –
  - If not specified, function definition is used for the declaration
  - Files much easier to read when top level function is at top of file
- C files typically contain:

  <function declarations>
  <top level function definition>
  ….
  <bottom level function definition>

Example of RightSide Up Code

```
int poly(int x,int c1, int c2, int c3);
int square(int x);

int main() {
        int x=7;
        if (poly(x)<9) return 1;
        return 0;
}
int poly(int x,int c1, int c2, int c3) {
        return square(x) + 2*x + 4;
}
int square(int x) { return x*x; }
```

Top level function: “main”

- Operating system invokes your “main” function
- OS passes two arguments to “main”: `int argc, char **argv`
  - argc – argument count – number of arguments passed to main
  - argv – argument vector – list of strings – one for each blank delimited token on the command line
  - For instance after: `./mycmd p1 p2`
    - argc = 3
    - argv[0] -> “/home/tbarten1/examples/xmp_args/mycmd”
    - argv[1] -> “p1”
    - Argv[2] -> “p2”

Return from main

- OS requires main to return an “int”
- Interpreted as the “return code” from your program
  - Indicates if your program worked or failed.
- By Convention
  - “0” – indicates that your program worked correctly
  - Any thing else indicates a problem!