An introduction to C programming

EECS 348: Software Engineering

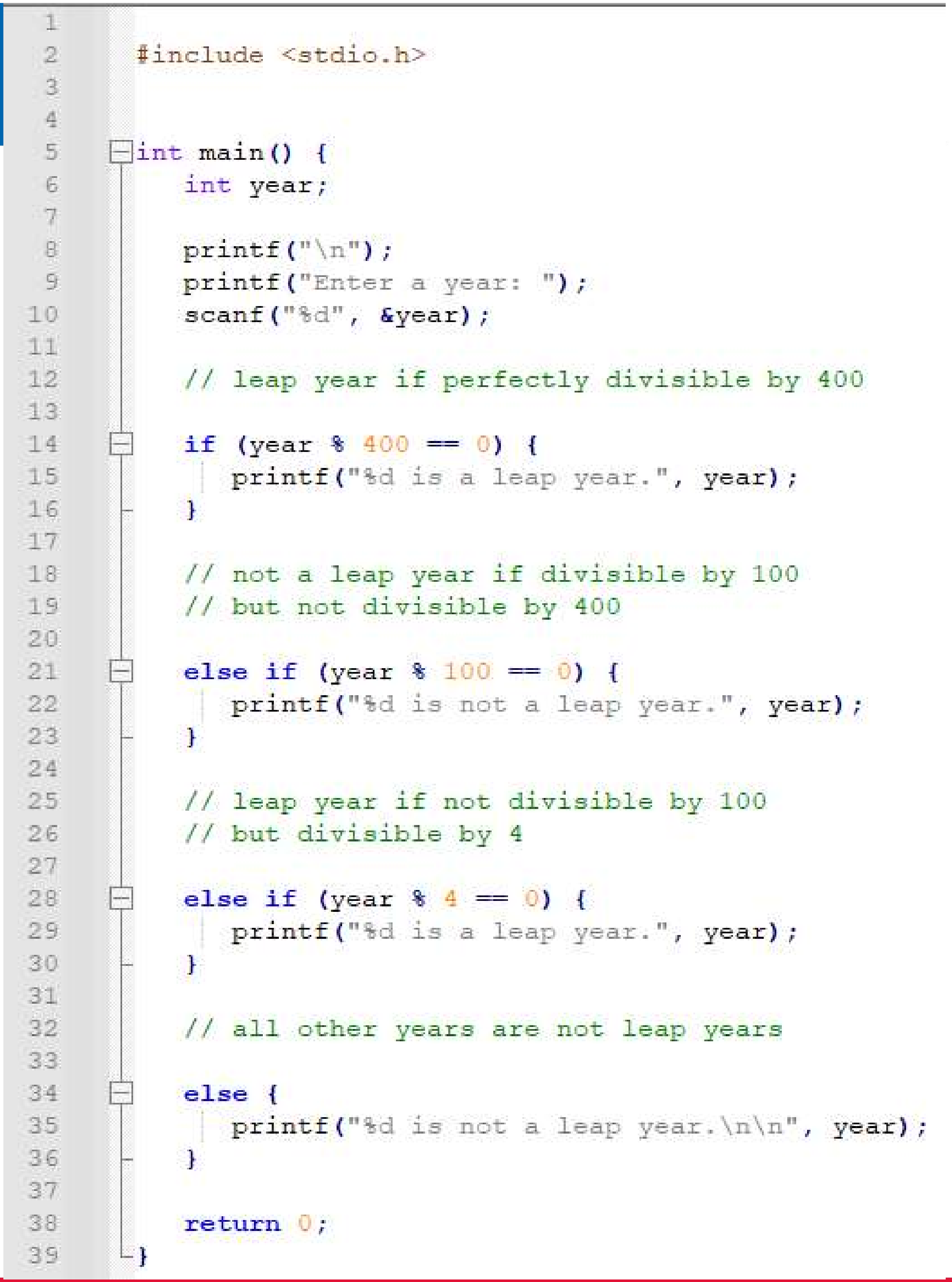
January 24, 2023

KU THE UNIVERSITY OF KANSAS

Learning objectives

- Learn how to write and compile a C program
- Learn what C libraries are
- Understand the C variable types
- Understand some control statements

A sample program



```

#include <stdio.h>


int main() {
    int year;

    printf("\n");
    printf("Enter a year: ");
    scanf("%d", &year);

    // leap year if perfectly divisible by 400

    if (year % 400 == 0) {
        printf("%d is a leap year.", year);
    }

    // not a leap year if divisible by 100
    // but not divisible by 400

    else if (year % 100 == 0) {
        printf("%d is not a leap year.", year);
    }

    // leap year if not divisible by 100
    // but divisible by 4

    else if (year % 4 == 0) {
        printf("%d is a leap year.", year);
    }

    // all other years are not leap years

    else {
        printf("%d is not a leap year.\n\n", year);
    }

    return 0;
}
```

Writing a C program

1. Write the code for a program (source code) using an editor such as vi or nano, save as file `my_pgm.c`

```
#include <studio.h>

int main ( ) {
    printf(“Hello, world!\n”);
}
```

Compiling a C program

2. Compile the program to convert program from source to an “executable” or “binary”:

```
$ gcc -o my_pgm.exe my_pgm.c
```

3. If the compiler produces any errors, fix them and recompile

Executing a C program

2. Once there are now programming errors and you have a n executable code, run it:

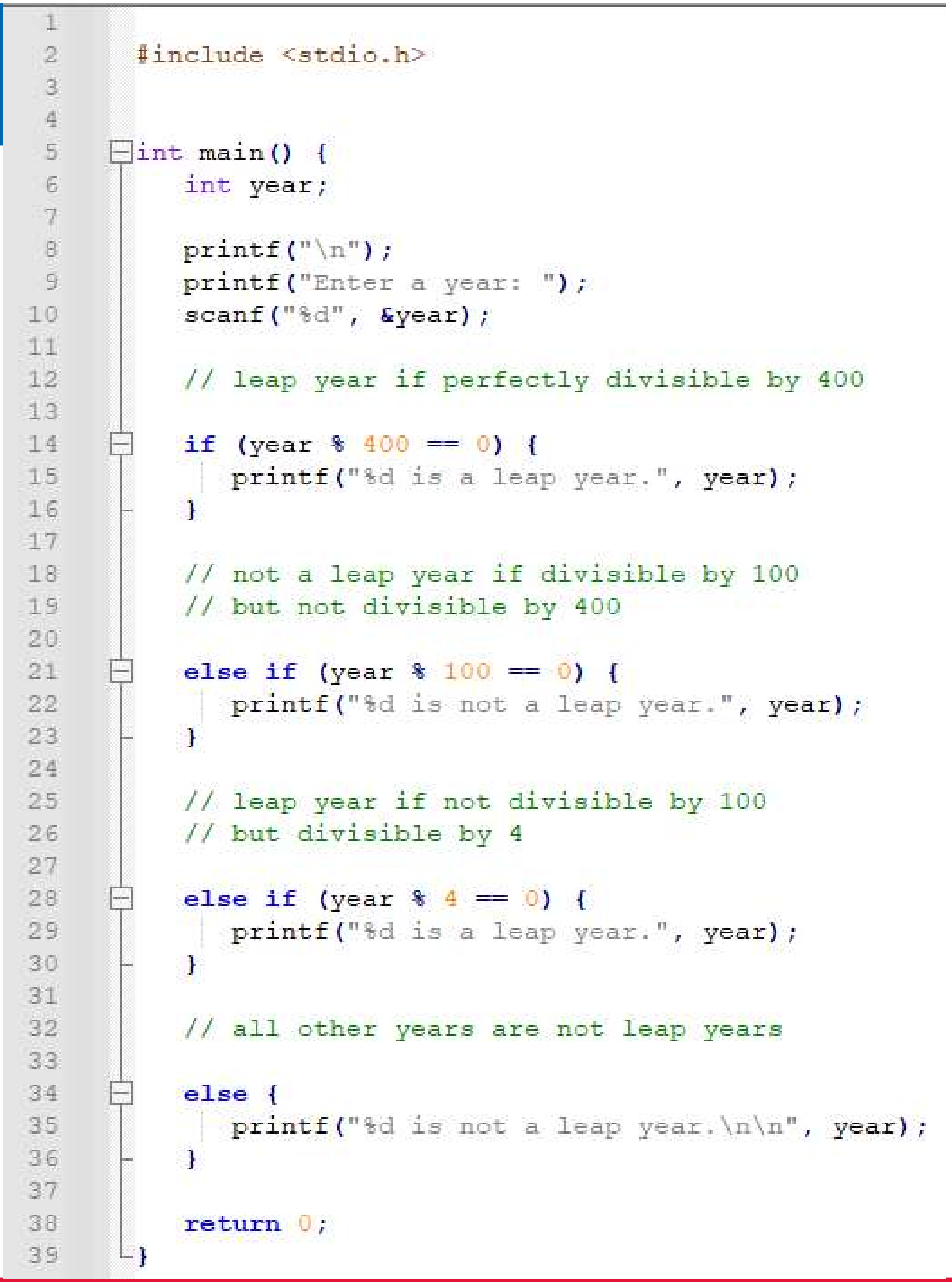
```
$ my_pgm.exe
Hello, world
```

Some common properties of C

- Case matters, white space does not
- Comments go between /* and */
- Each statement is followed by a semicolon
- Execution begins in the main function

```
#include <studio.h>
int main(int argc, char* argv[]) {
   /* start here */
   printf(“Hello World\n”);
   return 0;
   /*end here */
}
```

A sample program



```c

#include <stdio.h>


int main() {
    int year;

    printf("\n");
    printf("Enter a year: ");
    scanf("%d", &year);

    // leap year if perfectly divisible by 400

    if (year % 400 == 0) {
        printf("%d is a leap year.", year);
    }

    // not a leap year if divisible by 100
    // but not divisible by 400

    else if (year % 100 == 0) {
        printf("%d is not a leap year.", year);
    }

    // leap year if not divisible by 100
    // but divisible by 4

    else if (year % 4 == 0) {
        printf("%d is a leap year.", year);
    }

    // all other years are not leap years

    else {
        printf("%d is not a leap year.\n\n", year);
    }

    return 0;
}
```

Some common properties of C

What do the < > mean?

#include inserts another file. “.h” files are called “header” files. They contain stuff needed to interface to libraries and code in other “.c” files.

This is a comment. The compiler ignores this.

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

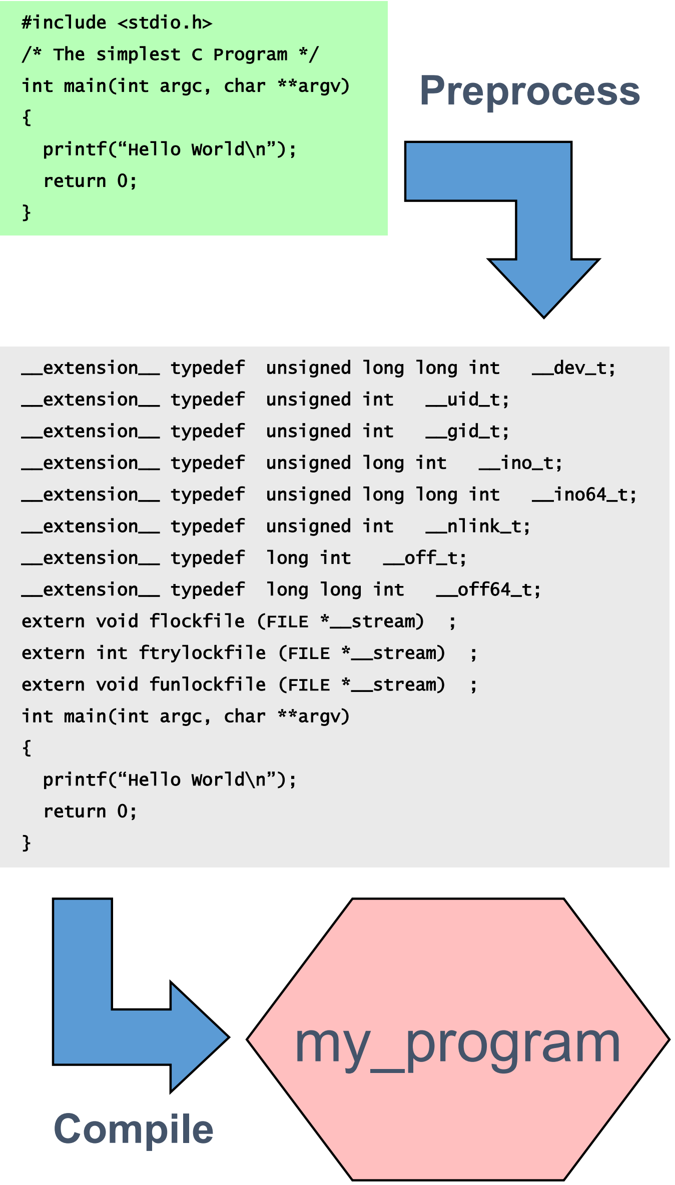
The main() function is always where your program starts running.

Blocks of code (“lexical scopes”) are marked by { … }

Return ‘0’ from this function

Print out a message. ‘\n’ means “new line”.

The compilation process



```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

```
__extension__ typedef  unsigned long long int   __dev_t;
__extension__ typedef  unsigned int   __uid_t;
__extension__ typedef  unsigned int   __gid_t;
__extension__ typedef  unsigned long int   __ino_t;
__extension__ typedef  unsigned long long int   __ino64_t;
__extension__ typedef  unsigned int   __nlink_t;
__extension__ typedef  long int   __off_t;
__extension__ typedef  long long int   __off64_t;
extern void flockfile (FILE *__stream)  ;
extern int ftrylockfile (FILE *__stream)  ;
extern void funlockfile (FILE *__stream)  ;
int main(int argc, char **argv)
{
  printf(“Hello World\n”);
  return 0;
}
```

Compilation occurs in two steps: “Preprocessing” and “Compiling”

Why ?

In Preprocessing, source code is “expanded” into a larger form that is simpler for the compiler to understand. Any line that starts with ‘#’ is a line that is interpreted by the Preprocessor.

- Include files are “pasted in” (#include)
- Macros are “expanded” (#define)
- Comments are stripped out (/*   */ , //)
- Continued lines are joined (\ )

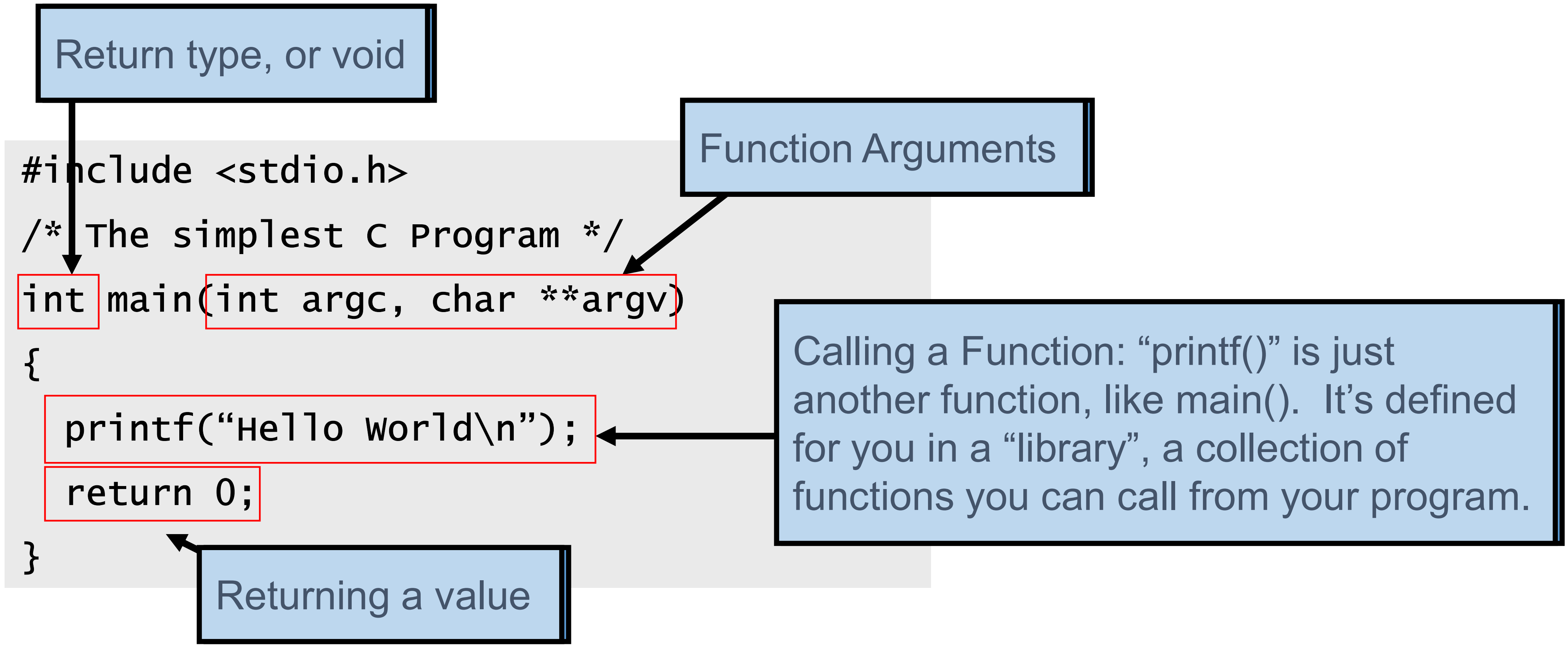
\ ?

The compiler then converts the resulting text into binary code the CPU can run directly.

C functions

A Function is a series of instructions to run. You pass Arguments to a function and it returns a Value.

"main()" is a Function. It's only special because it always gets called first when you run your program.



Return type, or void

Function Arguments

```
#include <stdio.h>
/* The simplest C Program */
int main(int argc, char **argv)
{
  printf("Hello World\n");
  return 0;
}
```

Calling a Function: "printf()" is just another function, like main(). It's defined for you in a "library", a collection of functions you can call from your program.

Returning a value

Memory locations

Memory is like a big table of numbered slots where bytes can be stored.

The number of a slot is its Address.
One byte Value can be stored in each slot.

Some “logical” data values span more than one slot, like the character string “Hello\n”

A Type names a logical meaning to a span of memory. Some simple types are:

`char`	a single character (1 slot)
`char [10]`	an array of 10 characters
`int`	signed 4 byte integer
`float`	4 byte floating point
`int64_t`	signed 8 byte integer

not always…

Signed?…

Addr	Value
0	
1	
2	
3	
4	‘H’ (72)
5	‘e’ (101)
6	‘l’ (108)
7	‘l’ (108)
8	‘o’ (111)
9	‘\n’ (10)
10	‘\0’ (0)
11	
12	

72?

What are C libraries?

- C is a lightweight language
  - Most of its intelligence is compartmentalized in libraries
  - Almost all c programs use the “stdio” or standard input/output library
  - Many also use the “math” library
- To use a library, include the header file (i.e., `stdio.h`) at the top of the file
- For most special purpose libraries (i.e., math) you need to include the math library

C variable types

- The most common types are: char, int, float, and double
- Strings are arrays of characters (we'll cover arrays later)
- Declare a variable before you use it:

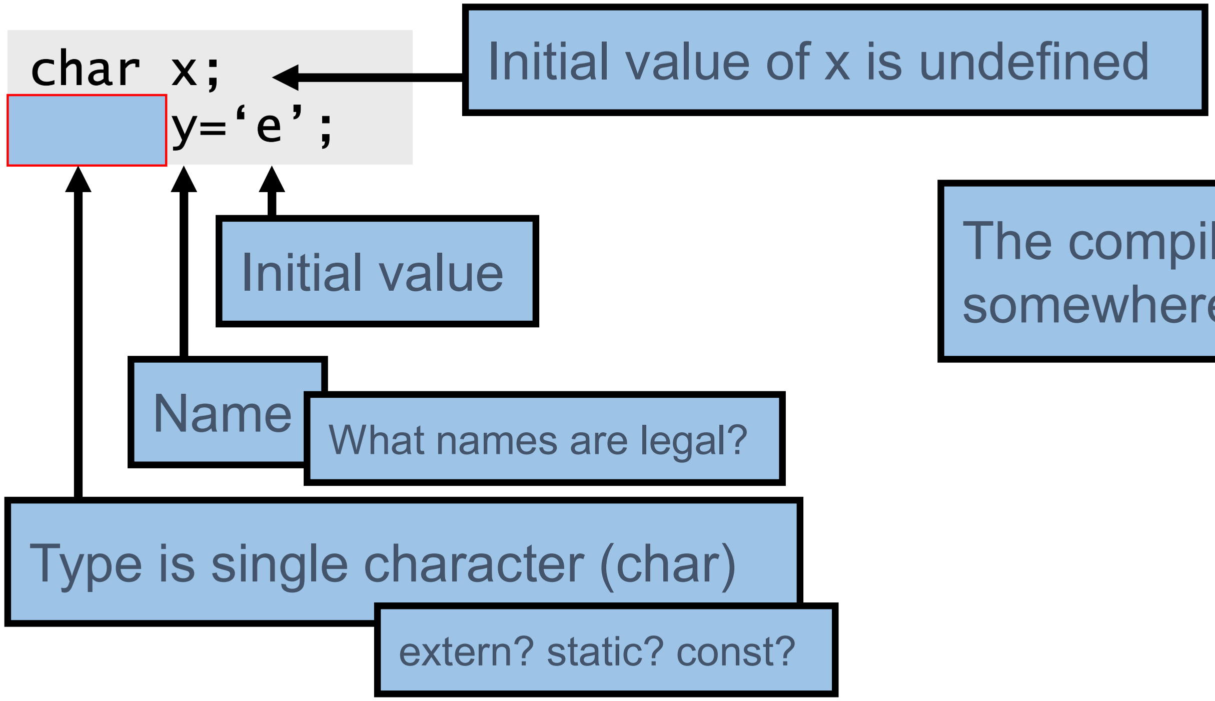
```
 /* declares an integer called x.  Its value is not assigned.*/
int x;
/* declares two floating point numbers;  set z equal to pi */
float y, z = 3.14159;
z = 4;  /* now z is equal to 4 */
myVal = 2;      /* An error because myVal is not declared. */
```

C variables

A Variable names a place in memory where you store a Value of a certain Type.

You first Define a variable by giving it a name and specifying the type, and optionally an initial value

declare vs define?



```
char x;
     y='e';
```

Initial value of x is undefined

Initial value

Name

What names are legal?

Type is single character (char)

extern? static? const?

The compiler puts them somewhere in memory.

symbol table?

Symbol	Addr	Value
	0	
	1	
	2	
	3	
x	4	?
y	5	'e' (101)
	6	
	7	
	8	
	9	
	10	
	11	
	12	

Expressions and evaluation

Expressions combine Values using Operators, according to precedence.

```
1 + 2 * 2       → 1 + 4       → 5
(1 + 2) * 2     → 3 * 2       → 6
```

Symbols are evaluated to their Values before being combined.

```
int x=1;
int y=2;
x + y * y       → x + 2 * 2     → x + 4       → 1 + 4       → 5
```

Comparison operators are used to compare values.
In C, 0 means “false”, and *any other value* means “true”.

```
int x=4;
(x < 5)                  → (4 < 5)                → <true>
(x < 4)                  → (4 < 4)                → 0
((x < 5) || (x < 4))     → (<true> || (x < 4))    → <true>
```

Not evaluated because first clause was true

Comparison operators

```
== equal to
<  less than
<= less than or equal
>  greater than
>= greater than or equal
!= not equal
&& logical and
|| logical or
!  logical not
```

```
+  plus
-  minus
*  mult
/  divide
%  modulo
```

```
&  bitwise and
|  bitwise or
^  bitwise xor
~  bitwise not
<< shift left
>> shift right
```

The rules of precedence are clearly defined but often difficult to remember or non-intuitive. When in doubt, add parentheses to make it explicit. For oft-confused cases, the compiler will give you a warning “Suggest parens around …” – do it!

Beware division:

- If second argument is integer, the result will be integer (rounded):
  5 / 10 → 0 *whereas* 5 / 10.0 → 0.5
- Division by 0 will cause a FPE

Don’t confuse & and &&..
1 & 2 → 0 *whereas* 1 && 2 → <true>

The if statement

- Syntax: if (expression) statement;
- If the expression is true (not zero), the statement is executed. If the expression is false, it is not executed.
- You can group multiple expressions together with braces:

```
if (expression) {
  statement 1;
  statement 2;
  statement 3;
}
```

The if/else statement

- Syntax: if (expression) statement1; else statement2;
- If the expression is true, statement1 will be executed, otherwise, statement2 will be

```
if (myVal < 3)
   printf(“myVal is less than 3.\n”);
else
   printf(“myVal is >= to 3.\n”);
```

Assignment operators

```
x = y    assign y to x
x++      post-increment x
++x      pre-increment x
x--      post-decrement x
--x      pre-decrement x
```

```
x += y   assign (x+y) to x
x -= y   assign (x-y) to x
x *= y   assign (x*y) to x
x /= y   assign (x/y) to x
x %= y   assign (x%y) to x
```

Note the difference between ++x and x++:

```
int x=5;
int y;
y = ++x;
/* x == 6, y == 6 */
```

```
int x=5;
int y;
y = x++;
/* x == 6, y == 5 */
```

Don't confuse = and ==! The compiler will warn "suggest parens".

```
int x=5;
if (x==6)   /* false */
{
  /* ... */
}
/* x is still 5 */
```

```
int x=5;
if (x=6)   /* always true */
{
  /* x is now 6 */
}
/* ... */
```

The “while” loop

- Syntax: while (condition) {statement;}
- The condition is evaluated, if it is true, the body of loop will be executed

```
while(condition){
      //code to be executed
}
```

The for loop

- Syntax: for (initialization; test; increment) {statements;}
- The for loop will first perform the initialization. Then, as long is test is TRUE, it will execute statements. After each execution, it will increment

```
for (i = 0; i < 3; i++) {
  printf(“Counter = %d\n”, i);
}
```

The “for” loop

The “for” loop is just shorthand for this “while” loop structure.

```
float pow(float x, uint exp)
{
  float result=1.0;
  int i;
  i=0;
  while (i < exp) {
    result = result * x;
    i++;
  }
  return result;
}

int main(int argc, char **argv)
{
  float p;
  p = pow(10.0, 5);
  printf(“p = %f\n”, p);
  return 0;
}
```

```
float pow(float x, uint exp)
{
  float result=1.0;
  int i;
  for (i=0; (i < exp); i++) {
    result = result * x;
  }
  return result;
}

int main(int argc, char **argv)
{
  float p;
  p = pow(10.0, 5);
  printf(“p = %f\n”, p);
  return 0;
}
```

Summary

- Learned how to write and compile a C program
- Learned what C libraries are
- Introduced the C variable types
- Introduced how to use if and if/else statements
- Introduced how to use the for and while statements

- References: some slides from Lewis Girod, CENS Systems Lab