FCPA 2022  
  
Memory, Data Types and Identifiers

Student Workbook 03

January 11, 2022

Author

Paul Kimball  
Interface Associates

1. An Appetizer -  
   Visual Studio Project Templates

# Make a Project Template

## Making a new C project in Visual Studio often starts the same way:

* + Create an empty project
  + Add a C file for the main program

#include <stdio.h>

write main() function

* + Set your favorite preprocessor config parameters:

#define \_CRT\_SECURE\_NO\_WARNINGS

* You can save yourself some work by creating a project template
  + Just create a simple project with your preferred files and settings, and export it as a template
  + The template will be available when you create a new project
* Project menu -> Export Template...
  + Brings up the export template wizard
  + Select "Project Template"
  + Fill in default project name and a description

1. Memory, Data types and Identifiers

Functions and Variables

* Variables hold values (called objects)
* Functions do things with the variables
* Libraries contain functions and variables written by others

**/\* This is a C program \*/**

**#include <stdio.h>**

**int square(int x){**

**return x \* x;**

**}**

**int main(void){**

**int value = 12;**

**int answer;**

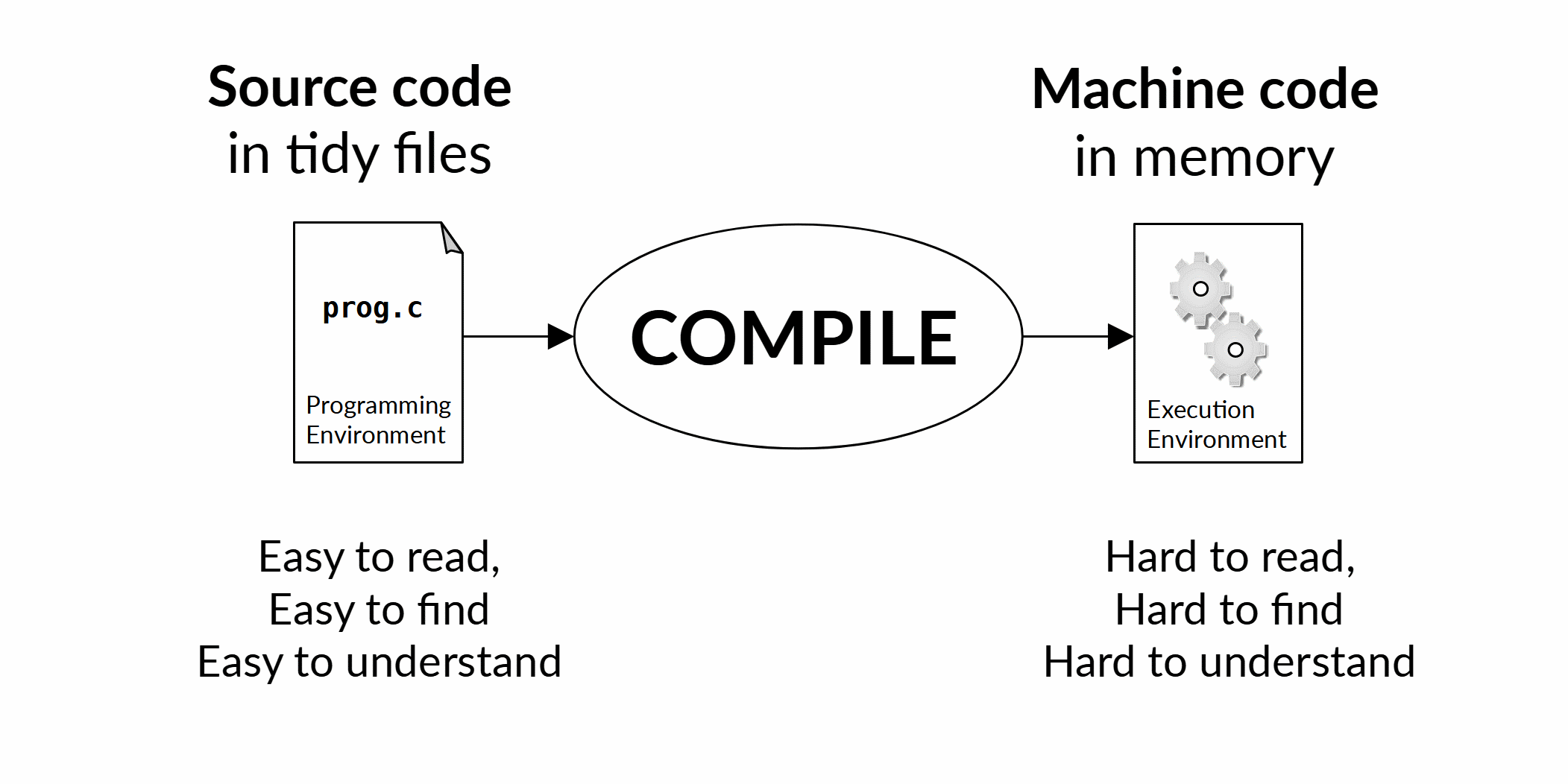
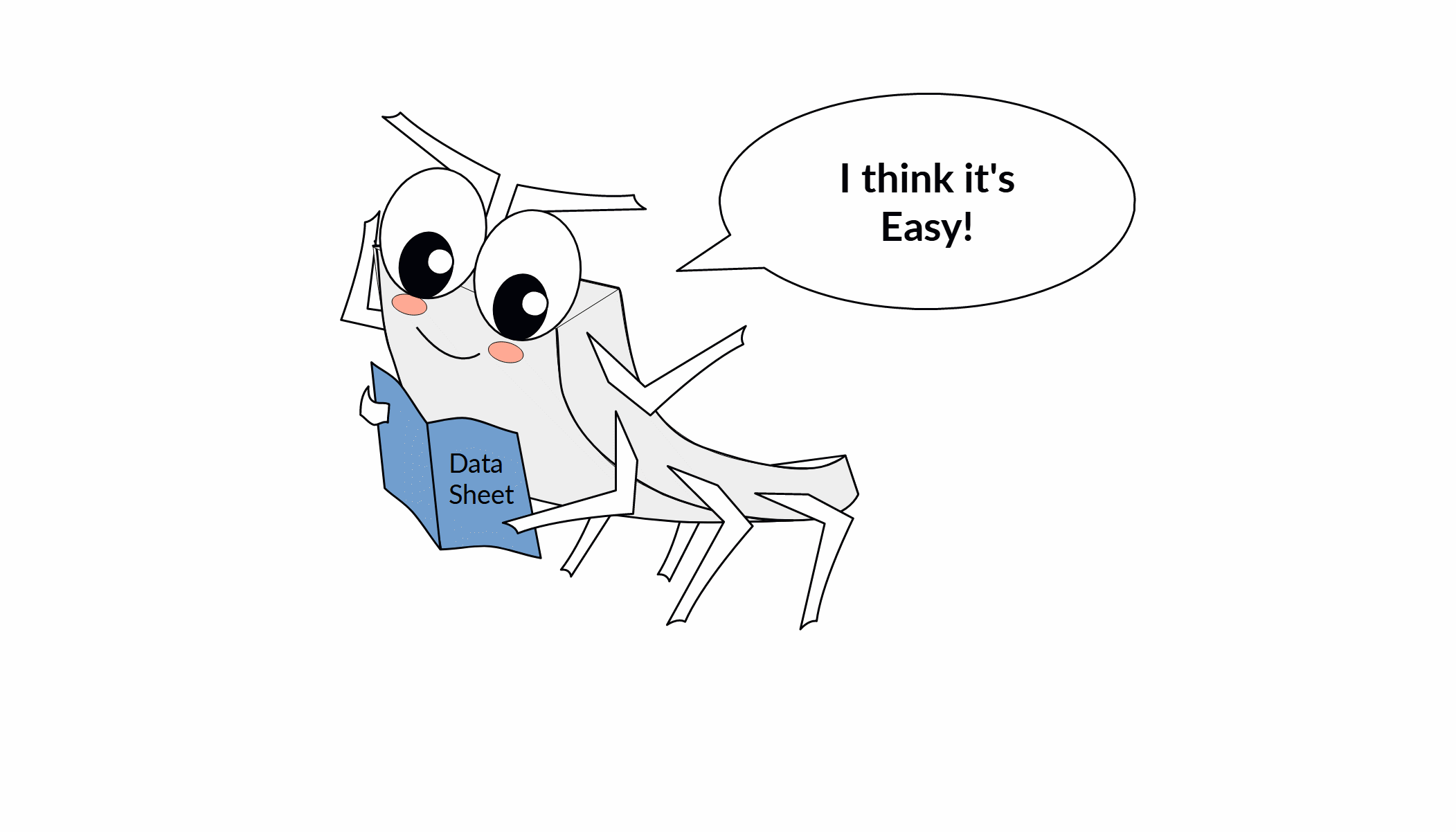
**answer = square(value);**

**printf("%d squared is %d\n", value, answer);**

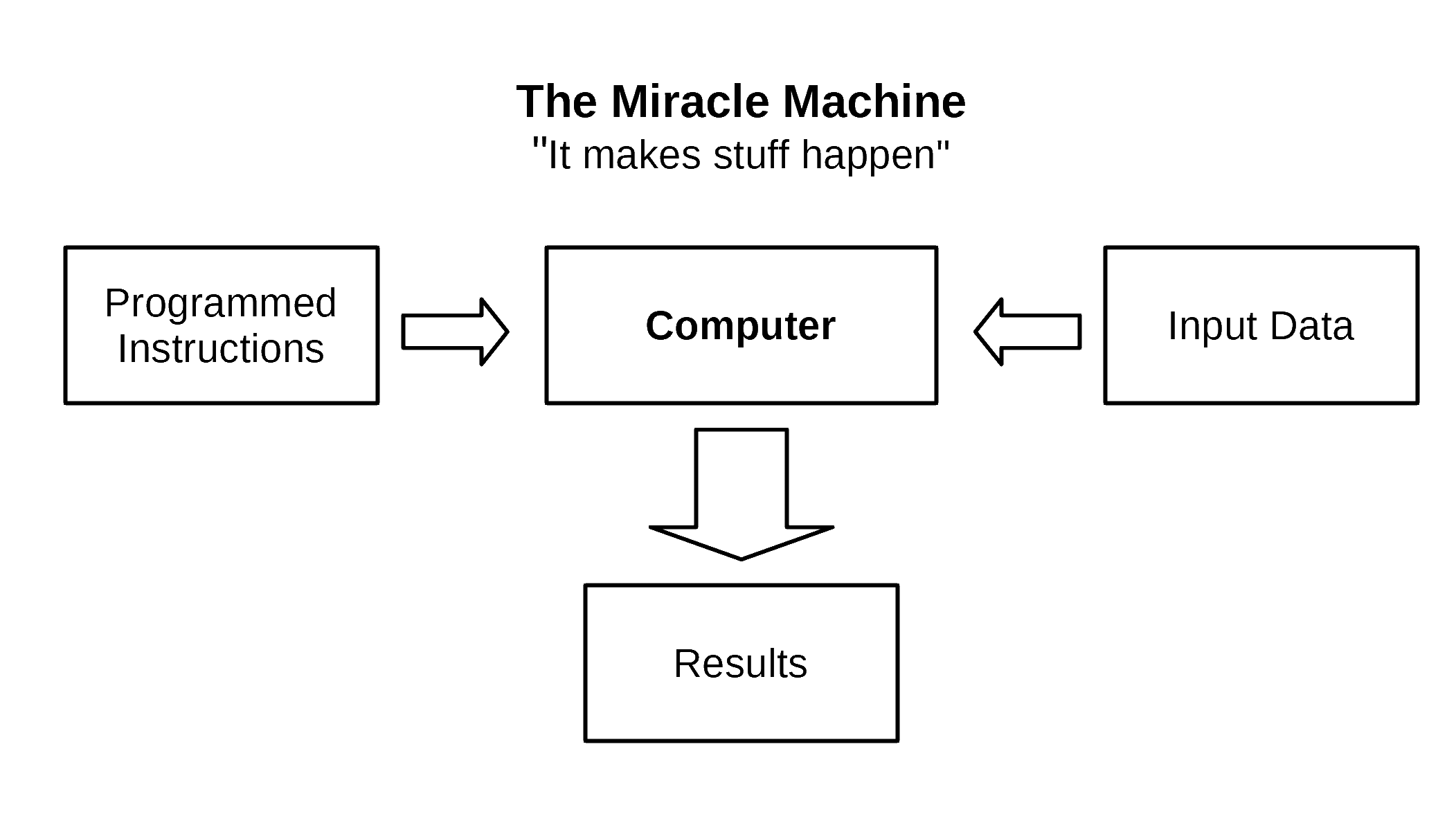
**return 0;**

**}**

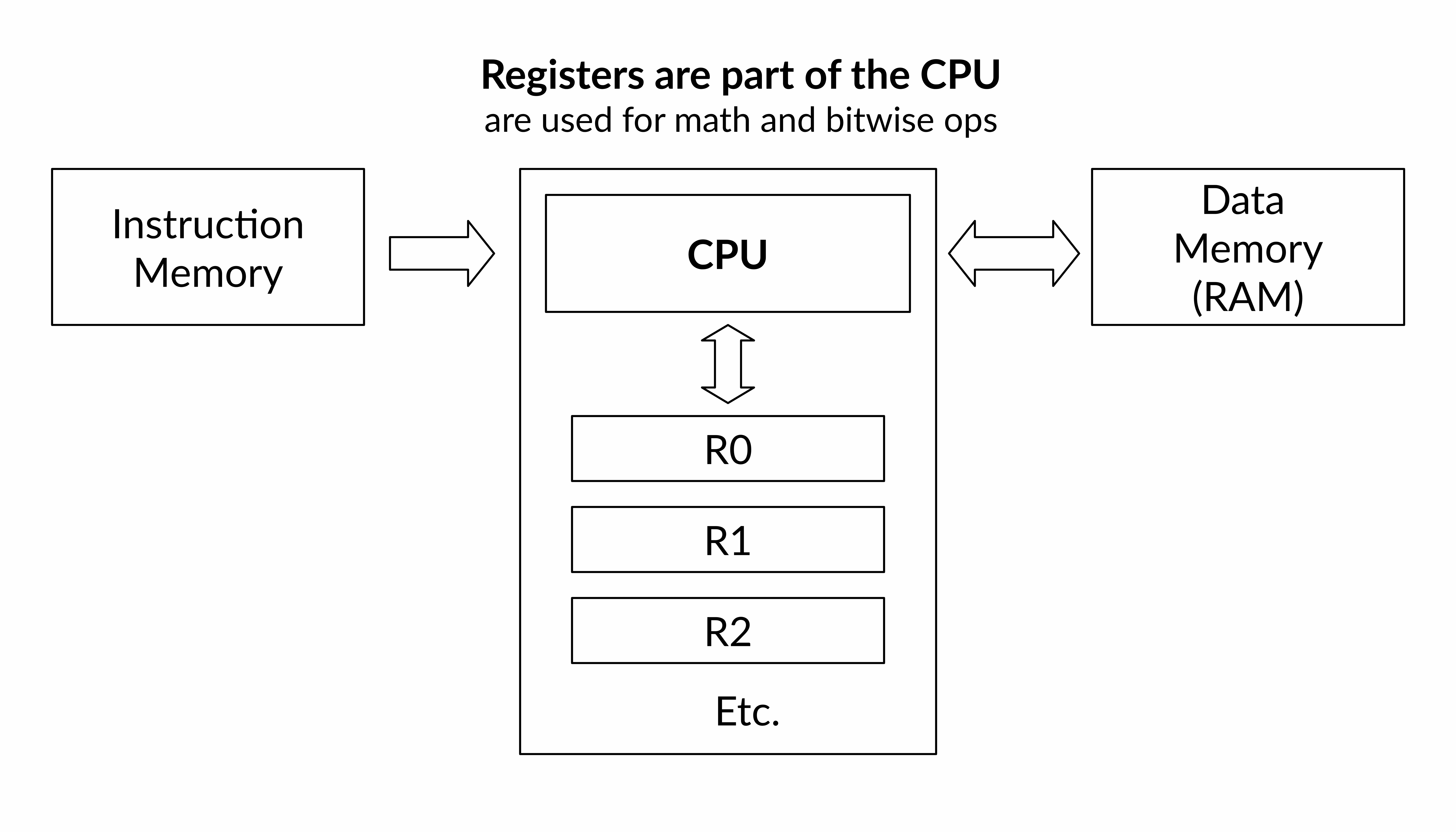
We Write Source Code



A Computer



Registers and Memory



* Registers are the fastest type of memory
  + Built into the CPU
  + Very Expensive
  + Not very many of them ( usually between 8 and 32 )
  + Not very big - ( usually between 8 and 64 bits )

Random Access Memory

* Instructions and data go in less-expensive memory outside the CPU proper



Memory in C

* *Memory* is composed of units of storage called "bytes"
  + *A byte is a unit of memory that is large enough to hold a single character in the execution character set*
  + *The number of bits in a byte is implementation-dependent (!)*

**See the value of CHAR\_BIT in the header file <limits.h> if you want to know the answer for your execution environment**

* Each byte has its own unique address
  + *An address is numeric value that references a particular location at run time*
  + *The number of bits in an address is implementation-dependent (!)*

#### **sizeof( void \*) will tell you the the answer (in bytes) for your execution environment**

* + Each byte can be read and written separately

But there may be a performance penalty for "unaligned" addresses

* The specification does NOT define
  + How much memory is available at execution time
  + Which addresses are valid and whether they are accessible
  + The CPU's preferred alignment and size of accesses, and whether there is a penalty for unaligned access

Object

* An *object* is a contiguous sequence of one or more bytes in memory
* An object represents a value of interest that should be understood as a unit
  + Your checking account balance
  + The mass of the Earth
  + The number PI
  + A list of lotto numbers
  + A cat
  + A concert ticket
* If you just look at the bytes by themselves, that is called the "object representation"
  + Good for storing the bytes or sending them across a network
  + Not very useful for most purposes

Data Type

* A *data type* describes how an object should be treated when used with operators and expressions
  + How much room it takes up in memory
  + Which operations are legal
  + Which operations make sense
  + What the result of an operation will be
* C has a modest set of data types
  + Arithmetic types

Integers, floating point, complex

* + Characters
  + Booleans
  + Pointers
  + Compound (structured) data types

structs, arrays, unions

* Problems we wrestle with in C
  + Some objects are too big to fit in a machine register

Have to be split up and recombined in various ways

* + Some data types are more expensive to manipulate than others

Floating point > integer

* + Some combinations of operators and types make no sense

But may be "legal" anyway

* + We usually just trust the compiler to deal with these problems

But we know it may cost us some performance

Exact Instructions Challenge



* The compiler is pretty smart, but...
  + ...you have to tell it everything! AND...
* You have to tell it before it needs it!

float x;

x = y \* 2;

float y = 15.; /\* oops! too late! Won't compile \*/

* + This is why header files are included *first*
  + That's why the main function shows up *last*

Identifiers

## We describe our world to the compiler by defining identifiers

## Everything gets a name

* + Variables
  + Functions
  + Data types

Declarations

* A *declaration* establishes an *identifier* and its *data type*
* A variable declaration
  + Data type says what kind of objects can be *stored*

**float** temperature\_F;

* A function declaration
  + Data type says what kind of objects are *returned*

**float** get\_engine\_temperature(){  
 return 220.0F;  
}

* A type declaration
  + Defines a new name for an existing data type

typedef **float** temperature\_type;

* Identifiers must be declared *before* they are used in code

# Rules for our identifiers

* Always case sensitive
  + HomeAddress and homeaddress are different identifiers
* Valid characters:
  + Alphabetic (uppercase and lowercase)
  + Numeric
  + Underscore
* Allowable names:
  + Cannot start with a number
  + Cannot contain punctuation, operators, special symbols
  + Must not conflict with a reserved name (see next page)
* Acceptable identifiers

add

add\_all

m\_length

RADIUS

input\_file\_29

SupportTicket

* Not OK

\_ItsMyParty

and\_I'll\_cry

if-i-want-to

\_CRT\_SECURE\_NO\_WARNING

Reserved Names

## C Language keywords

* + These are predefined by the language itself
  + In a couple weeks, you'll know them all

|  |  |  |  |
| --- | --- | --- | --- |
| **auto** | **extern** | **short** | **while** |
| **break** | **float** | **signed** | **\_Alignas** |
| **case** | **for** | **sizeof** | **\_Alignof** |
| **char** | **goto** | **static** | **\_Atomic** |
| **const** | **if** | **struct** | **\_Bool** |
| **continue** | **inline** | **switch** | **\_Complex** |
| **default** | **int** | **typedef** | **\_Generic** |
| **do** | **long** | **union** | **\_Imaginary** |
| **double** | **register** | **unsigned** | **\_Noreturn** |
| **else** | **restrict** | **void** | **\_Static\_assert** |
| **enum** | **return** | **volatile** | **\_Thread\_local** |

## "Special" names (reserved for the compiler and tools)

* + Names that start with two underscores
  + Names that start with underscore followed by an uppercase letter

#### This technically allows names that start with an underscore followed by a *lowercase* letter

* Function and Macro names defined in the standard libraries (if the associated header file is included)
  + We'll see these later when we look at the libraries

Identifier Scope

* Once an identifier has been declared, it can be used in code
  + But not everywhere!
* The *scope* of an identifier defines a contiguous region where it can be used
  + Starts at the point of declaration!
* Variables declared outside a function have *file scope* - they are shared between functions
  + Scope terminates at end of translation unit
  + Visible in any files #included after the aration
* Variables declared inside a function have block scope - for local variables and function parameters
  + Scope terminates at end of enclosing block
  + Blocks can be nested
* Identifiers with the same name can be declared in different scopes
  + BUT, an inner-scoped identifier *hides* the outer-scoped identifier
  + Do this only if it is absolutely clear what you are referring to