SWINBURNE UNIVERSITY OF TECHNOLOGY

Introduction to Programming

Week 11,

Other Languages

KNOWING

The C Programming Language

- A statically typed language

  (i.e variables must be declared to be of a particular type, and can only hold that type)
- A compiled language
- A functional language that supports structured programming.
- Developed by Dennis Richie 1971-73.

Dennis Richie

Richie was also heavily involved in the development of the Unix operating system.

Ritchie's death did not receive much news coverage as the media was focussed on Steve Jobs who died the week before.

Computer historian Ceruzzi stated that:[

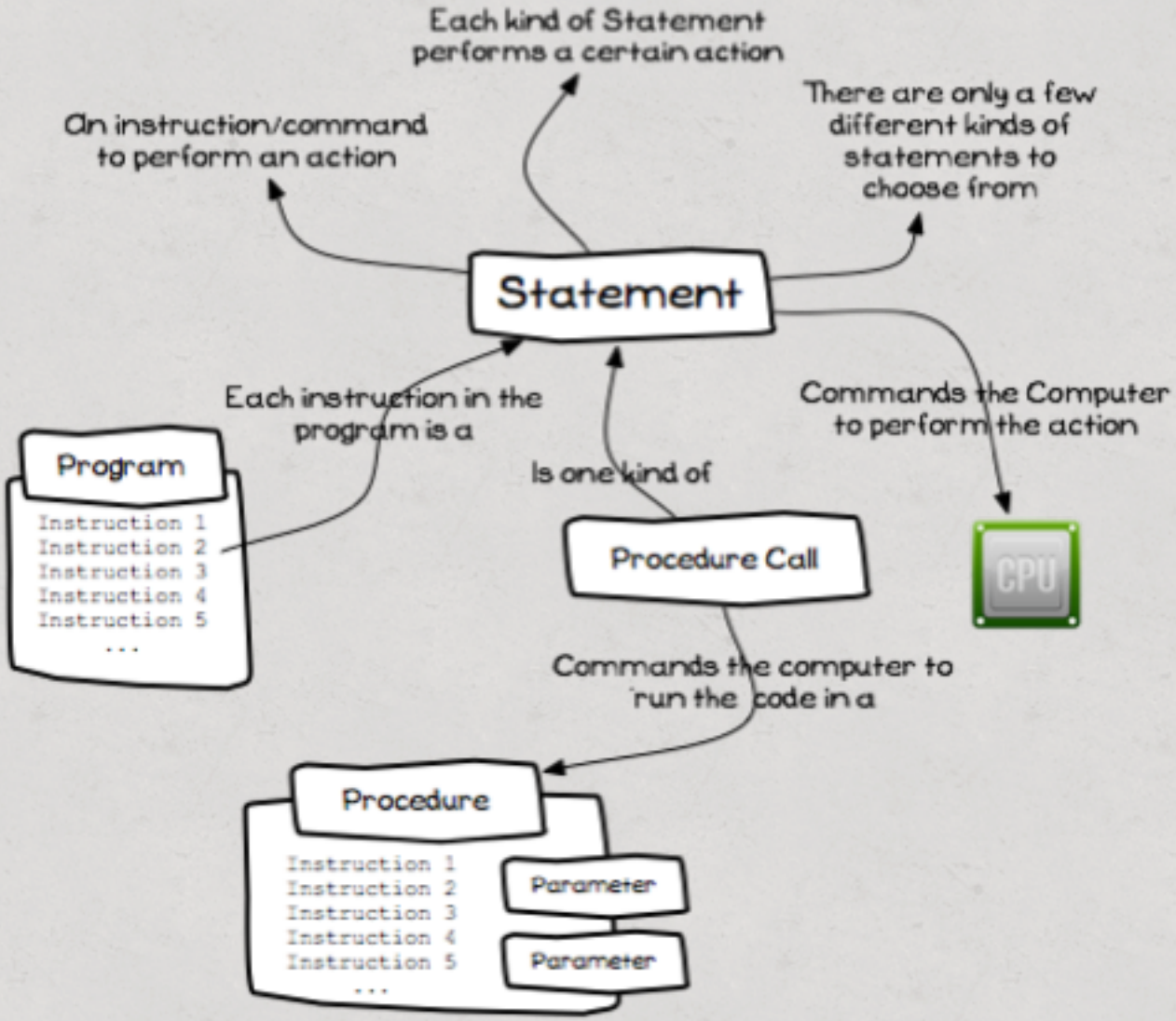
“Ritchie was under the radar. His name was not a household name at all, but [...] if you had a microscope and could look in a computer, you'd see his work everywhere inside. “

Sources: *Srinivasan, Rajeev (October 25, 2011).*
"Dennis Ritchie, a tech genius as great as Steve Jobs". Firstpost. Retrieved December 4, 2017.
Langer, Emily (October 14, 2011). "Dennis Ritchie, founder of Unix and C, dies at 70". Washington Post. Retrieved November 3, 2011.

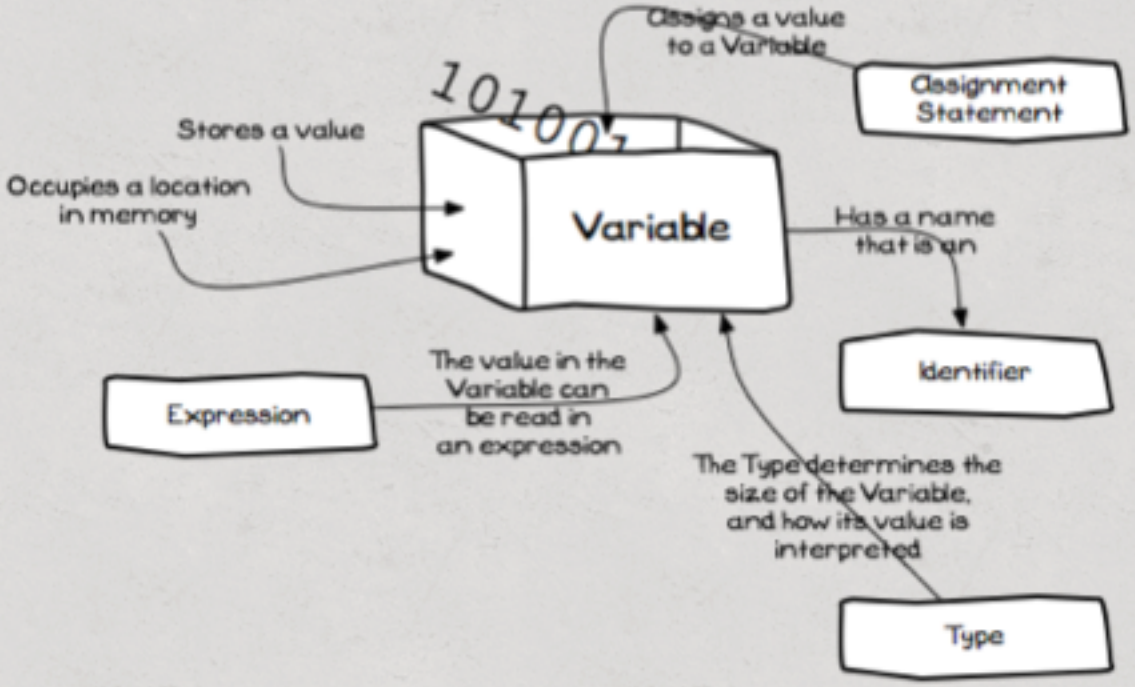
Learning other languages

- The concepts you have already learned are largely the same for other languages.
  - Variables and constants
  - Loops and conditional statements
  - Arrays to hold multiple values
  - Library files for code that is already written and can be re-used.

Languages have similar statements to perform actions



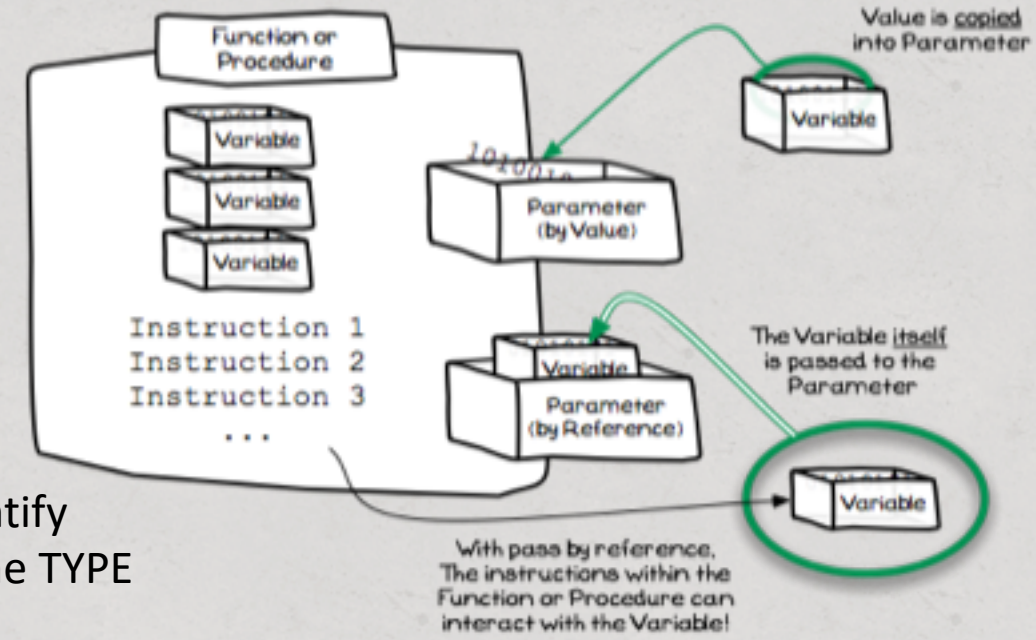
Variables and Constants are used to store values



But in C you must explicitly identify the TYPE of the variable. Eg:

```
char color[256];
char model[256];
int year;
```

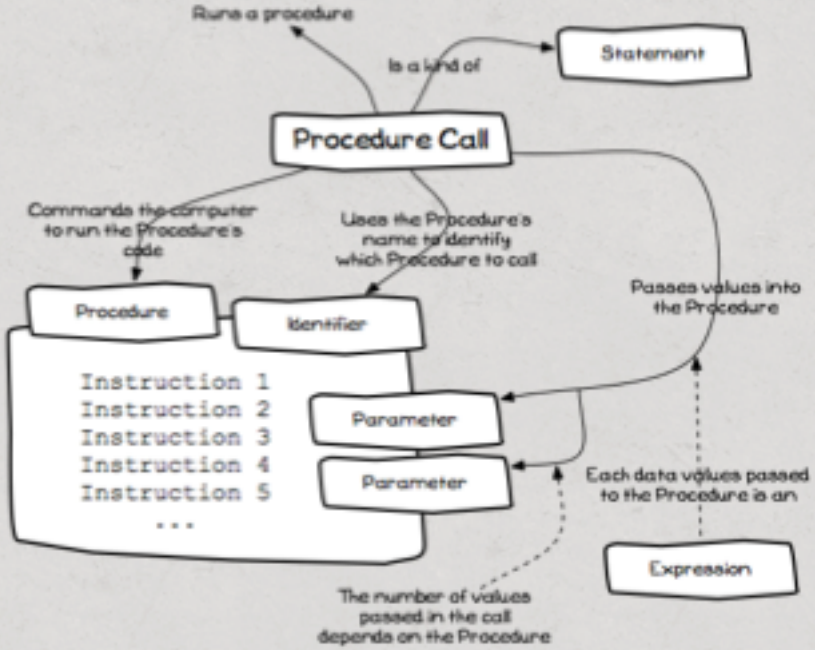
Code is divided into Functions and Procedures, with Parameters and Local Variables



But in C you must explicitly identify the TYPE of the function and the TYPE of the parameters. Eg:

```
void populate_car_array(car_data data[], int size)
```

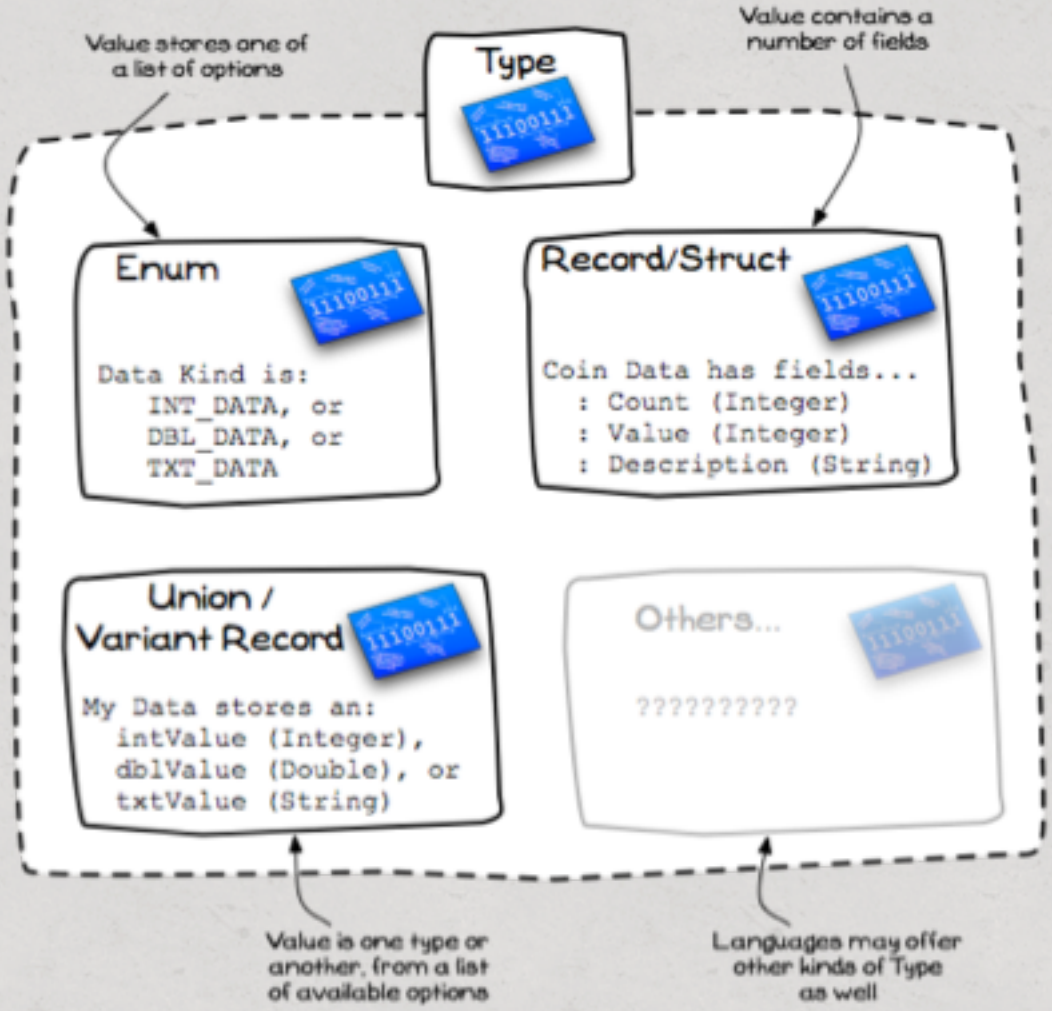
Use procedure calls to run procedures



Eg:

```
void print_car(car_data car)
{
  printf("\nCar Colour: %s", car.color);
  printf("\nCar Model: %s", car.model);
  printf("\nCar year: %d", car.year);
}
```

Create custom data types to organise your data

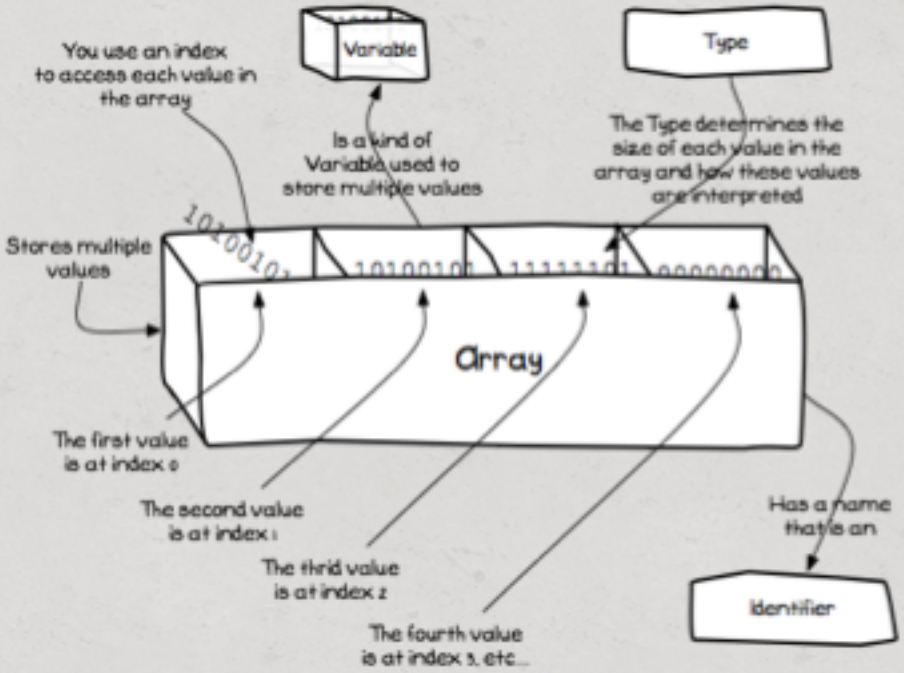


Eg:

```
typedef struct car_data
{
  char color[256];
  char model[256];
  int year;
} car_data;

car_data read_car()
```

Arrays are used to store multiple values



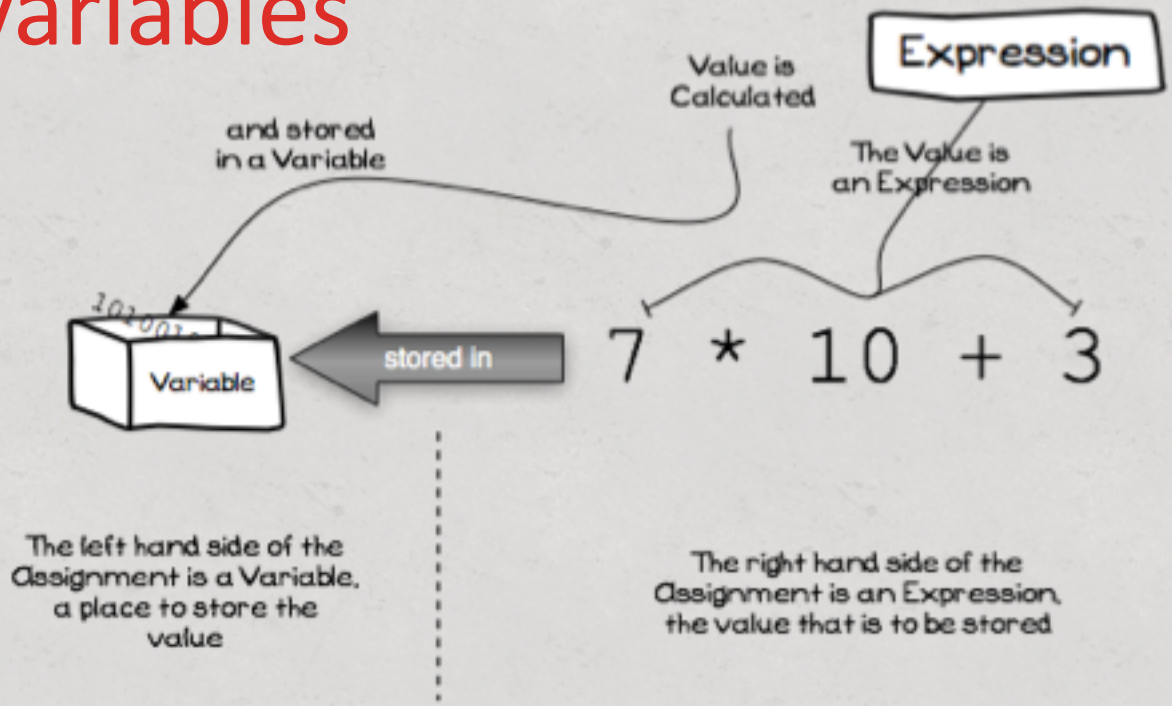
Eg:

```
int main()
{
  car_data cars[CAR_SIZE];

  populate_car_array(cars, CAR_SIZE);
  print_car_array(cars, CAR_SIZE);

  return 0; // exit... with result 0
}
```

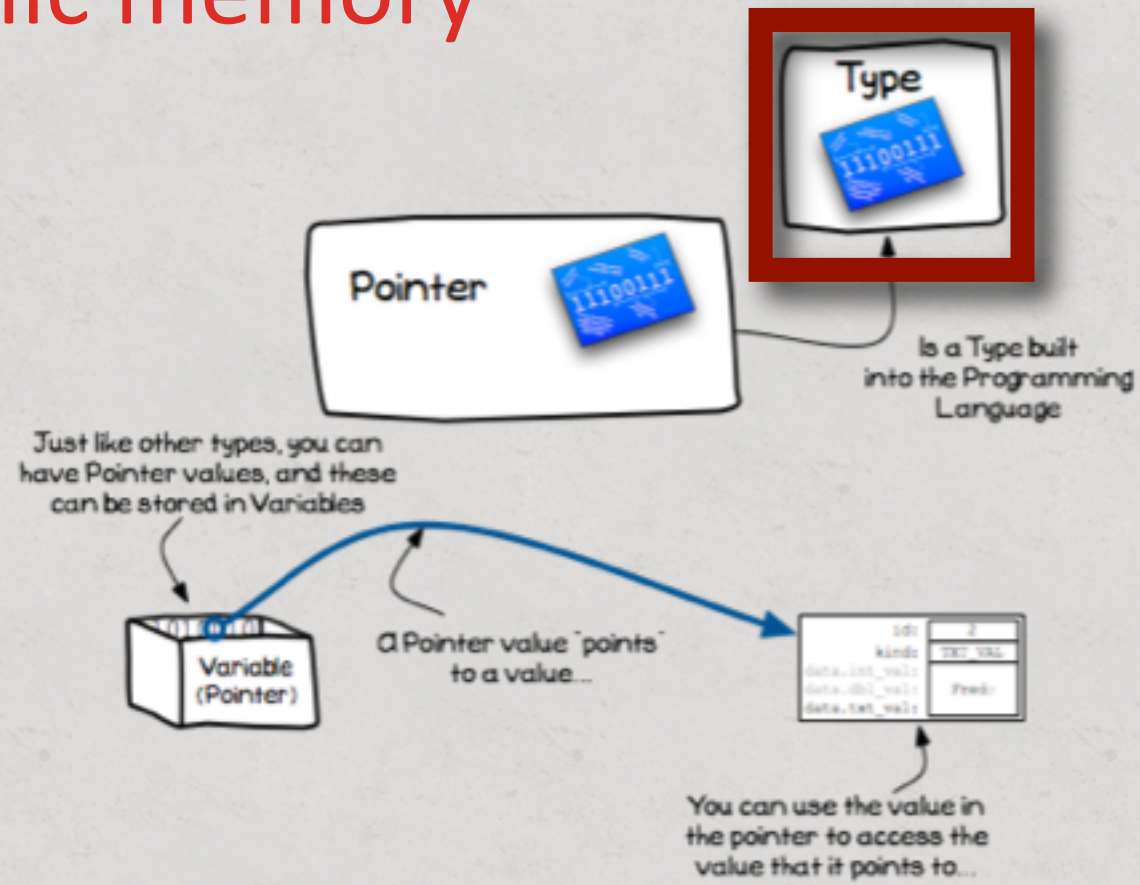
Use assignment statements to store values in variables



Eg:

```
for(i = 0; i < size; i++)
{
  data[i] = read_car();
}
```

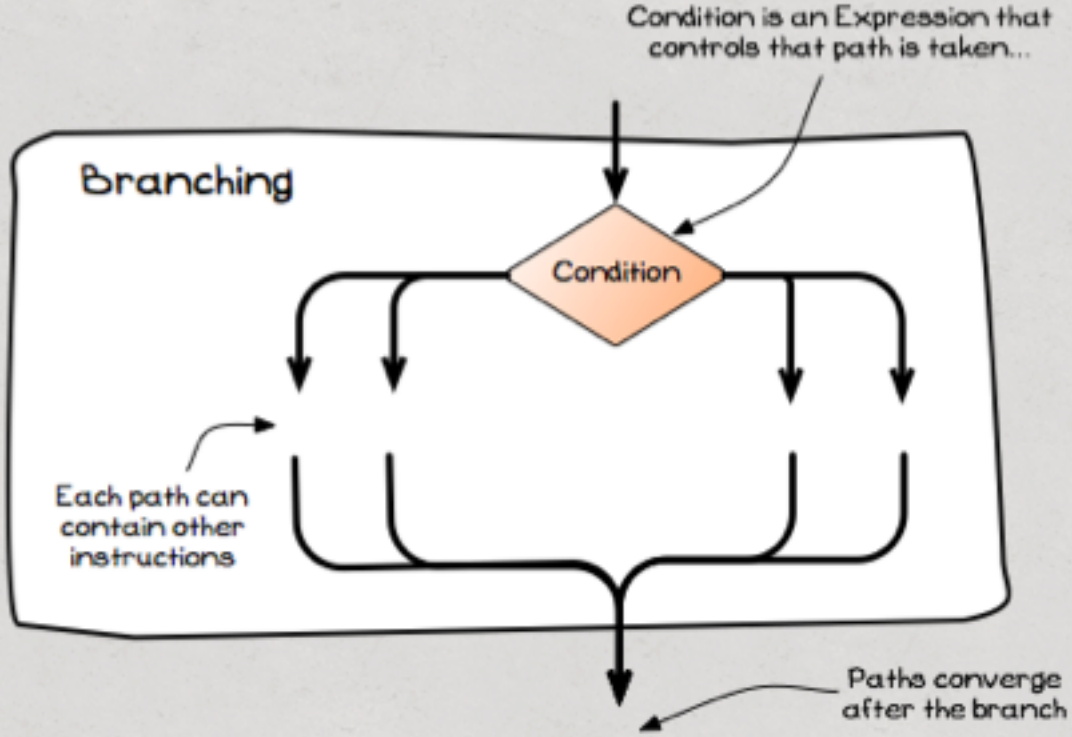
C uses pointers to work with relations and dynamic memory



Eg:

```
my_string read_string(const char* prompt);
```

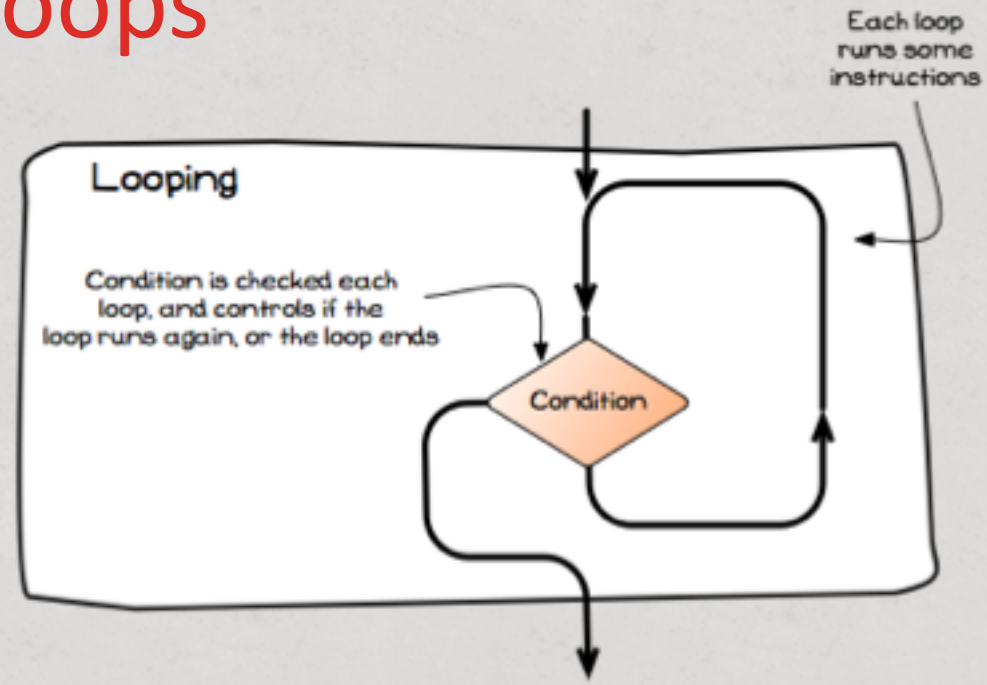
Branch with **if** and **case** statements



Eg:

```
if (size == 0) {
 printf("No elements to print - finished!");
 return;
}
```

Loop with **while**, **for**, and **repeat** or **do … while** loops



Eg:

```
void populate_car_array(car_data data[], int size)
{
  int i;

  for(i = 0; i < size; i++)
  {
    data[i] = read_car();
  }
}
```

```
index = 0;
while (index < size) {
  printf("\n Index is: %d", index);
  index ++;
}
```

Compiling C

- C programs are parsed by a pre-processor.
- This happens when you compile the program:

```
MacBook-Pro:Resources mmitchell$ gcc -o my_program read_write_record.c
```

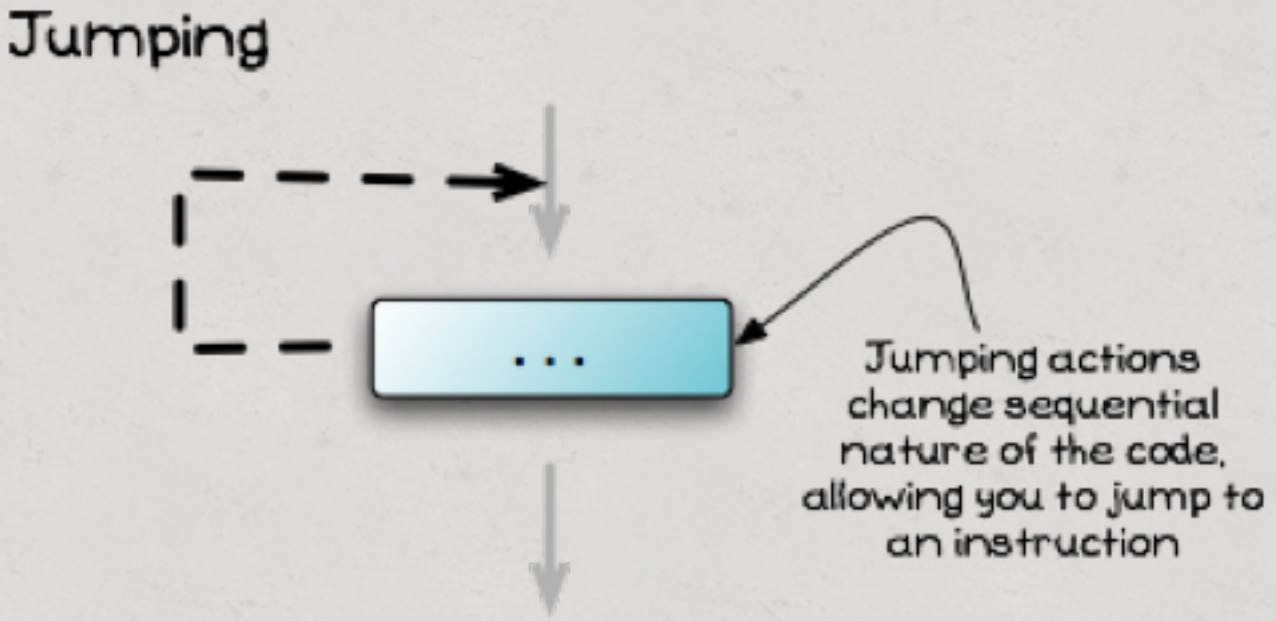
- The pre-processor pulls in any included libraries.
- The output is an executable file for the machine it was compiled for (i.e unlike Ruby, you cannot just a copy the program to another machine and run it, you need to compile it for each machine)
- You run the code by typing the executable file name at the terminal.

Eg:

```
MacBook-Pro:Resources mmitchell$ ./my_program
```

(a.out is the default name of the executable file)

Return or **Exit** from functions and procedures, and



Eg:

```
if (size > MAX) {
  printf("Too many elements in the file - aborting read.");
  return 0;
}
```

Using libraries

- In C you need to #include libraries (like **require** in Ruby)

```
#include <stdio.h>
```

Or:

```
#include "terminal_user_input.h"
```

- Ruby equivalent:

```
require './input_functions'
```

(although the base Ruby language has gets() puts() and printf() etc – which C does not)

Header Files

C has header files that contain function prototypes:
Eg: ***terminal_user_input.h***:

```
int read_integer(const char* prompt);

//
// Reads a integer from the user in a given range.
//
int read_integer_range(const char* prompt, int min, int max);

//
// Reads a double from the user.
//
double read_double(const char* prompt);
```

The actual functions are then contained in the matching
.c program file Eg: an extract from ***terminal_user_input.c***:

```
int read_integer_range(const char* prompt, int min, int max)
{
  int result = read_integer(prompt);
  while ( result < min || result > max )
  {
    printf("Please enter a number between %d and %d\n", min, max);
    result = read_integer(prompt);
  }
  return result;
}
```

Strings

- Not all languages have the same level of type support

Advanced String Support

```
name :=  “\(firstName) \(lastName)”;
```

Swift

Basic String Support

```
name :=  firstName + ‘ ‘ + lastName;
```

Pascal

Strings?

```
name = (char*) malloc(strlen(first_name) + strlen(last_name) + 2);

sprintf(name, “%s %s”, first_name, last_name);

free(name); //when done
```

C

string.h library

In C you need to use a function to compare strings (as they just an array of characters – the *strcmp* function checks each character to see if it is the same in both arrays)

Eg:

```
strcmp(name.str, "YOUR_TUTOR_NAME")
```

If the arrays have the same content, *strcmp* will return 0, otherwise it will return a positive integer.

You will need the above for this week's tutorial task.

Using C

- C is a language that may be described as “closer to the machine”:
  - It does less for you (so you can only have it do what you want)
  - But it generates smaller code that therefore runs faster (tradeoff productivity for performance)
  - It can appear cryptic, and gives more cryptic errors some other languages.

C and other languages

- C **syntax** is the basis for many modern languages
  - So knowing C is useful just for syntax.
- We use the GNU C++ compiler (gcc)
- So in labs you will need to run **mingw**.

Some Differences

- C is case sensitive
- You lose some features:
  - There are no Strings in C - you have to use arrays of characters
  - You cant pass by reference the way you do in Ruby (C++ can) - you have to pass pointers
  - Functions / procedures can't be passed or return arrays by-value (i.e a copy of the array)
    - To pass an array in you pass in a pointer to the 1st element

C String handling is lower level than Ruby

string = char* or char myArray[n]

- Arrays of characters, terminated with the null character
- Make sure you don't read/write beyond the end of the array
- In C you need to store the length of the array yourself (Ruby does it for you)

C Naming Conventions

- There are a number of C language conventions... we use...
  - lower_case = identifiers for functions/procedures, types, variables
  - UPPER_CASE = constants
  - Indent within { }, and control structures as before

Terminal and File Output

```c
#include <stdio.h>

void main() {

  int i = 9;
  char* s = "my string";

  // Print a string

  printf("my string");

  // Print an integer

  printf("%d", i);

  // More strings:
  printf("my string with a newline\n");

    // Integer and a string:

  printf("An integer: %d, and a string with a newline: %s\n", i, s);

}
```

Use printf() to write to the terminal

(and use fprintf() to write to file)

printf is defined in the library <stdio.h>

(<*somelib*.h> means a system library)

Terminal and File Input

- Read from the user with our functions for read string etc. Use **scanf()** of **gets()** to read values from the user
- Both printf and scanf use format strings to indicate the format of the output/input data
- *You could use the supplied terminal user input code — (access the string characters using the* ***.str*** *field of the record/struct)*

Example input and ouput in C

```c
#include <stdio.h>

void main() {

  int i = 9;
  char* s = "my string";
  char s2[256];

  // Print a string
  printf("my string");

  // Print an integer
   printf("%d", i);

  // Print a string with a newline
  printf("my string with a newline\n");

  // Print an integer and a string:
  printf("An integer: %d, and a string with a newline: %s\n", i, s);

  printf("\nEnter a string:");
  gets(s2); // read a short string

  printf("\nEnter an integer:");
  scanf("%d", &i); // read an integer

  printf("Read in %d, and %s2", i, s2);
  fflush(stdout);

}
```

NB: *gets()* is deprecated in C, the alternative is in the demo code for this week.

Summary

- We have looked at the C language
- The concepts should be the same, although the syntax and implementation of somethings (eg: Strings) can be different
- Use the example code above to help with this week's tutorial task.
- Lets see some more examples.
  - Eg: Talk through then run the demo code for this week:

```
gcc democode.c terminal_user_input.c -o run
```