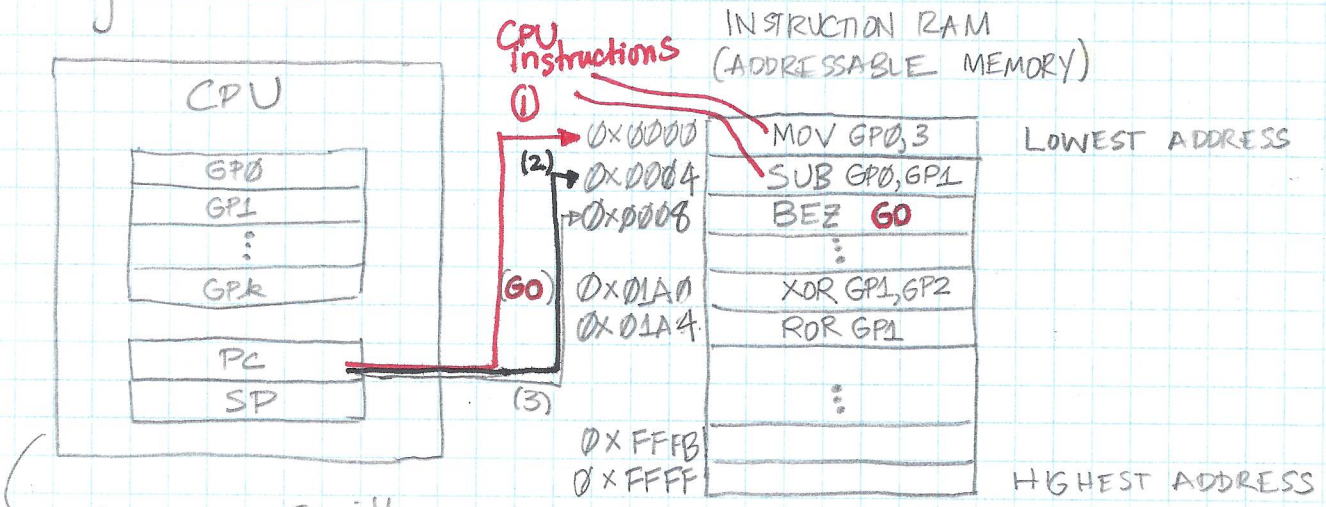
INTRODUCTION

WHAT IS A COMPUTER PROGRAM?

- in a sense, a computer program is a "script" performed by a microprocessor; modern digital computers, based on finite state machines, were conceived in order to be general purpose devices, capable of acting on a set of instructions given to them by human users.

- an understanding of computer programs comes from an understanding of how microprocessors work; a microprocessor contains various registers, some for general-purpose use, and others that have a specific function, such as the stack pointer (SP) and program counter (PC) registers:



microprocessor with registers shown; recall that a CPU has an instruction set from which programs can be built

- The CPU performs the first instruction at the address pointed to by the PC register at (1)
- When the instruction at 0x0000 completes, the PC points to the next instruction at (2)
- Again once the second instruction is completed the PC moves to the next instruction (3)
- Note that BRANCHING can occur, as with the "branch if equal to zero" instruction at address 0x0008.

- if the result of "SUB GP0, GP1" was zero, the PC is loaded with the address 0x01A0 (corresponding to the "GO" label in the program)

- from this example we can think of a computer program as a series of instructions with a (default) sequential flow of logic that may also involve branching (i.e., movement of the PC that is not sequential).

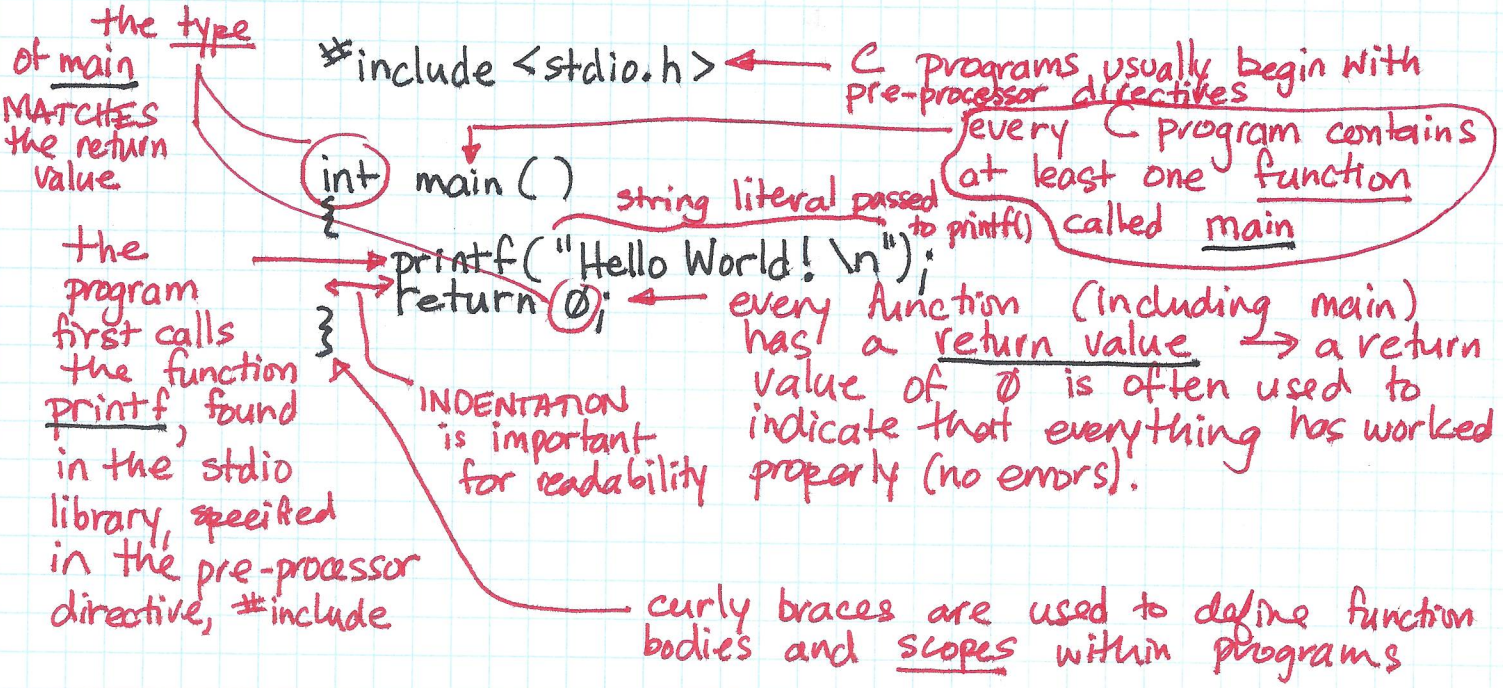
- the instruction set allows us to program a computer using assembly language, however for large programs which need portability assembly language is challenging to work with, requiring that the program be re-written if it must run on a different microprocessor architecture.

- It is because of the tedious nature of assembly language that developers have created high-level languages, which allow us to program computers without having to deal with the specifics of the underlying CPU architecture.

- C was developed in the late 1960s and early 1970s as a language for developing operating systems which could run on multiple computer platforms; the UNIX kernel, for example, was written in C and subsequently became popular because of the widespread adoption made possible by the portability of the UNIX software. Today, billions of devices run variants of UNIX, in the form of Android OS, Linux or Mac OS.

AN INTRODUCTION TO C

- Consider the following "Hello World!" C program:



- enter this program, build it, and execute it in your development environment

SHORTEST POSSIBLE C PROGRAM: (with no compile-time errors or warnings) `void main(){}`

variables

- programs need variables to store, process and evaluate data
- in C, variables have definite types; the basic types in C include:

`int`	(integer type)
`char`	(character type) ← for single characters
`float`	(floating-point type) ← for numbers with fractional components
`void`	(a generic type, e.g., when no value is needed)

- note: it is possible to store string literals (like "Hello World! \n" above) in variables, in a pointer type, `char*`.

- here is a slightly expanded Hello-World program:
  (please see the latest version on the course Git repository")

```
#include <stdio.h>

/* global variables */

char    message[50] = "---";
char    user_name[30];

int main()
{
    /* local variables */

    int mood;
    char pets;

    /* request user's name */
    printf("Please enter your name: ");
    scanf("%s", user_name);

    /* display message */
    message = "Hello! It is nice to meet you, ";
    printf("%s", message);
    printf("%s!\n", user_name);

    /* ask questions */
    printf("\nI hope you do not mind answering some questions!\n");

    printf("On a scale of 1 to 10, how would you rate your mood today? (please enter a number): ");

    scanf("%d", &mood);

    printf("\n Do you have any pets? (please enter Y or N)? ");

    scanf("%c", &pets);

    printf("\n\n Thank you for answering my questions, %s, goodbye!", user_name);

    /* exit successfully */
    return 0;
}
```

global variables: can be used any where in the program

local variables: defined only within the scope in which they are defined – in C, scopes are usually explicitly defined with { ... }

another function in stdio, which retrieves input from the "standard input", in this case, the computer keyboard

format specifier (string literal)

string literal

~~string literals are terminated with a null character~~

- enter, build and execute this program

- this program is more interesting than the Hello-World program, but it is a little disappointing. The program collects data, but does little with it
- let's expand the program so that its response changes, depending on the user's responses.
- for example, if the user gives a low mood value, the response could be encouraging; if the user's mood is good, the program could acknowledge that.
- in C, we can use a special control structure to make decisions. It is called an if statement. Consider the following code fragment:

```
char *mood_response;
mood_response = "That's Wonderful!"; /* default response */
if (mood < 6)
{
    mood_response = "I am sorry, I hope you feel better soon, ";
}
printf(mood_response);
printf(" %s!\n", user_name);
```

- insert this code fragment into your program, and test the results.

- the if statement also has an "else" option to expand its capabilities:

```
if (...)
{
    :
}
else
{
    :
}
```

thus we could rewrite the above fragment as

```
char *mood_response
if (mood >= 7)
{
    mood_response = "That's Wonderful\0";
}
else
{
    mood_response = "I am sorry, I hope
        you feel better soon\0";
}
printf ...
```

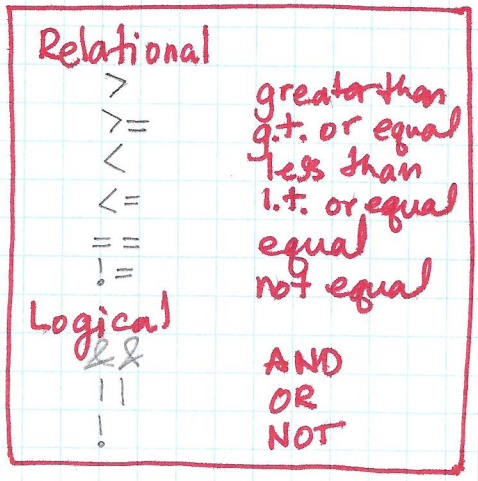
- try out this version of the program. Do you notice any differences?

- the if-else statement can be laddered to allow even greater decision-making ability:

```
if (...)
{
    :
}
else if (...)
{
    :
}
else if (...)
{
    :
}
    :
else
{
    :
}
```

in C, true literally equals 1 and false is equal to Ø.

Please note! TRUE OR FALSE (1) (Ø)



- For example:

```
char *mood_response
if (mood < 3)
{
    mood_response = "Oh that's dreadful. Sorry, ";
}
else if((mood == 3) || (mood < 7))
{
    mood_response = "I am sorry, feel better soon, ";
}
else /* mood must be > 7 */
{
    mood_response = "That's Wonderful, ";
}
```

logical OR

- try out this code block, or a variation!

- HOMEWORK: produce an expanded version of the hello_1 program that takes into account whether the user has pets, as follows

- if the user has a mood ≤ 3, the program responds with:

  "Maybe cuddling with your pet will help!" if the user has a pet, and

  "Studies show that pets can improve our overall sense of well-being. Why not get one now?!" if the user does not have a pet.

- if the user has a mood ≥ 4, but < 7, the program responds with:

  "I am sure spending time with your pet will cheer you up!" if the user has a pet and

  "All you need is to spend time with a new friend. A visit to the animal shelter is the ticket!" if the user does not have a pet.

- if the user has a mood ≥ 7 the response should be:

  "Great news! Celebrate with your pet!" if the user has a pet and

  "That's great! Imagine if could just share your happiness with a new friend. Please visit your local animal shelter!" if the user does not have a pet.