Advanced_C_Programming

This repo contains c advanced topics on various topics, its purpose is to provide educational and useful content to people with codes collected from different sources.

All of the explanations of codes will be explained in readme.

Please take a look to *README.pdf* for best view.

using_header_files_effectively

This include functions that can update date, finding date of tomorrow etc.

This folder include 2 files, main.c and date.h

date.h include the functions prototypes,enumed variables and macro will be used in main.c

First let's explain date.h

```
#include <stdbool.h>
```

This include the stdbool library this library defines the bool type and the true and false constants

```
enum KMonth {January = 1, February, March, April, May,
             June, July, August, September, October, November, December};
```

```
enum KDay {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
```

enum: Codes the keywords to integers, in `January = 1` , 1 is the first element of coding and after this January keyword always equal to 1, but in KDay when no number is specified the coding start with `Sunday = 0`.

```
typedef struct date{
    enum KMonth month;
    enum KDay day;
    int year;
}Date;
```

Date is a data structure that include month as an object of Kmonth, day as an object of Kday, and year as an integer.

```
Date date_update(Date today);
int numberOfDays(Date d);
bool isLeapYear(Date d);
```

Those are the prototypes of wrtied maked in the c file.

```
#define setDate(s, mm, dd, yyyy) s = (Date) {mm, dd, yyyy}
```

setDate is a macro function that can assign month day and year to the structure given.

```
extern Date today;
```

This block pf code is n example to extern a variable from the main script as an example main.c

Secondly main.c

`bool isLeapYear(Date d)` function return True if the year is leap false if not.

`int numberOfDays(Date d)` function return the number of days in the current month.

`Date date_update(Date today)` function using the previous functions return the date of tomorrow.

storage_classes

Automatic variables

Take a look to automatic_variables.c

Automatic variables is the same with local variables are stored in stack memory. These allocates memory automatically when the function is called and frees the memory automatically when the function is returned.It's possible to use auto keyword to declare automatic variables but it's not necessary because all local variables are automatic variables by default.

Example usage with auto keyword:

```
#include <stdio.h>
int main(){
    auto int a = 10;
    printf("a = %d\n", a);
    return 0;
}
```

External variables

External variables are stored in data segment of memory, they are accessible from any function in the program. They are also called global variables. You can use `extern` keyword to get a global variable and use from other file. ***Please take a look to external_variables.c and external_variables1.c** *. They are initialized as 0.

Static variables

Take a look to static_variables.c

Static Local Variables

Static variables are stored in data segment of memory not stack,they are accessible only from the function they are declared in. They have the property of preserving their value

even after they are out of their scope. Hence, static variables preserve their previous value in their previous scope and are not initialized again in the new scope!!

```
#include <stdio.h>
int counter(){
    static int count = 0;
    count++;
    return count;
}
int main(){

    printf("%d\n", counter());
    printf("%d\n", counter());
    printf("%d\n", counter());
    return 0;
}
```

In this code when we call counter function, count variable is initialized to 0 and incremented by 1 . And if you run the code like I explain before you can see clearly that count variable is not initialized again when we call counter function again and just increments.

The static variables are initialized to 0 like global variables and they can only be initalized using constant literalse. In this example you cannot acess to count inside the while loop by extern or others method.

Static global variables

Take a look to ***External variables.** In that topic we get a variable by `extern` from another scop. If we change it with `static int x= 10;` this variable will be not accessible from the other scope.

Static functions

As an example we can define a static function as follow:

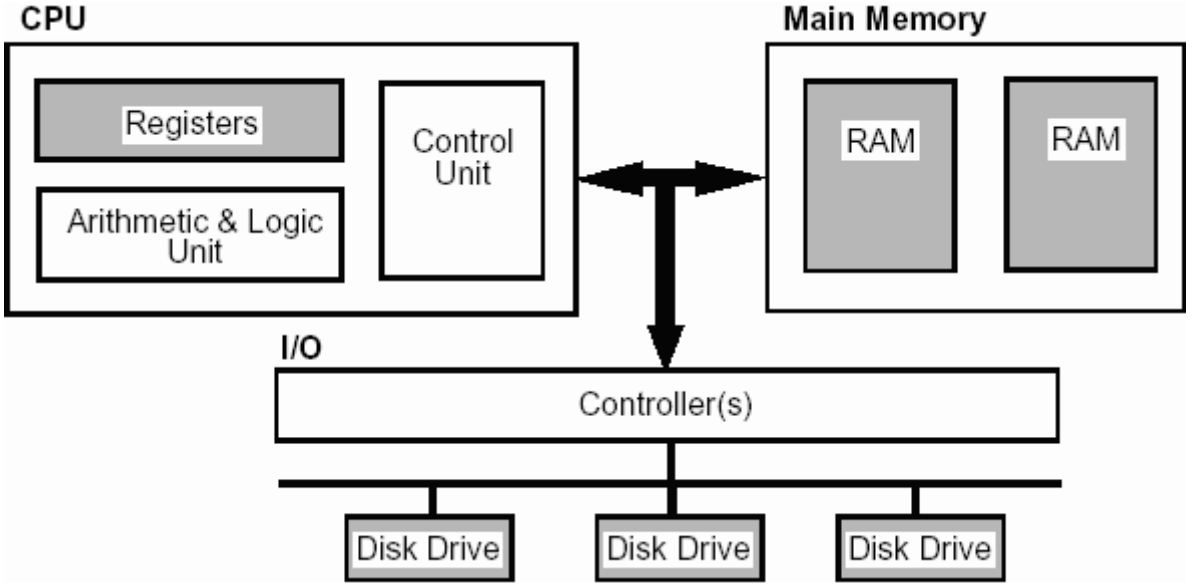
```
static void function(){
    printf("This is a static function\n");
}
```

Like static global variables this function is not accessible from other scope when include the file.

Register Variables

Take a look to register_variables.c

Registers hold operands or instructions that CPU would be currently processing. Memory holds instructions and the data about the currently executing program required by the CPU.



Since registers are quicker to access compared to memory, the `register` keyword can be employed in a C program to place variables that are frequently used into registers.

1. Using the & operator with a register variable could lead to an error or warning from the compiler (depending on the specific compiler in use). This is due to the fact that when a variable is declared as a register, it might be stored in a register instead of memory, making the act of accessing its address invalid. As an example:

```
void test1(){
    //Creating a register variable and accessing its address
    //is invalid because register variables are stored in CPU
    //Compiler will throw an error : error: address of register
variable requested
    register int i = 10;
    int* a = &i;
    printf("%d", *a);


}
```

2. The `register` keyword is applicable to pointer variables as well. It's evident that a register can hold the address of a memory location. The following program should

not encounter any issues.

```
void test2(){
    int j = 19;
    register int* a = &j;
    printf("%d", *a);

}
```

3. The usage of the `register` keyword is restricted to block scope (local) and cannot extend to the global scope (outside of the `main` function).

4. Modern compilers are sophisticated and often perform their own optimizations, including register allocation. They may choose to store variables in registers even without the explicit use of the `register` keyword.