202018021
Fu, Ziyu
Programming I Chapter 6 Task 1, 2
2021/06/10

Task 1

1. Objective
   To understand the mechanism between global and local variables using the sample code.

2. Strategy of solving
   - Read and understand the given sample code
   - Made changes according to the task statement
   - Run and observe the output

3. Program code

```c
#include <stdio.h>
void pr_local(void);
void pr_global(int i);
int i = 20;
int main(){
    int i = 0;
    for (i = 0; i < 2; i++){
        pr_local();
        pr_global(i);
        printf("i-main1: %d \n", i);
    }
    printf("i-main2: %d \n", i);
}
void pr_local(void){
    static int i = 0;
    i = i + 1;
    printf("i-local: %d \n", i);
}
void pr_global(int i){
    i = i + 1;
    printf("i-global: %d \n", i);
}
```

4. Results and discussions
   Console output:

```
i-local: 1
i-global: 1
i-main1: 0
i-local: 2
i-global: 2
i-main1: 1
i-main2: 2
```

The content of `main()` repeats two times, with `i-main1 = 0` and `i-main1 =1`.

In the first iteration, `i-main1 = 0`:

`pr_local()` declares a `static` variable `i` local to the function, and initializes it with a value of 0. Since it's a `static` variable, it's going to preserve its value in memory as long as this program runs. `i` is then incremented by 1, and then printed out. This function outputs 1.

`pr_global()` takes the `i` declared in `main()`, which was passed to `pr_global()` as an argument, and increments it by 1. It always outputs 1 higher than the `i` in `main()`, which in this case is 0+1=1.

In the second iteration:

`i` in `pr_local()` preserves its last value, which is 1, so it becomes 2 after incrementation.

`pr_global()` again takes the `i` in `main()` and increments by 1, which in this case is 2.

After the for loop, the last value of `i` in `main()` is printed. Since `i++` is post-increment, in the last iteration of the `for` loop, the old value of `i` was evaluated against the condition, and then incremented. So `i = 1` was tested to indeed satisfy `i < 2`, and then the value of `i` was incremented to 2. `i = 2` was then tested against `i < 2` but didn't satisfy, which breaks the `for` loop and `i = 2` becomes the last value of `i`.

Task 2

1. Objective
   To create a function that calculates the product between two 2*2 matrices, and store the value in a global variable in the file.

2. Strategy of solving
   - Understand the calculation of matrix multiplication, and the mechanism of global variables
   - Design the code
   - Observe the outputs

3. Program code

```c
#include <stdio.h>

void calculate(void);

int a[2][2] = {5, 0, 8, 1};
int b[2][2] = {1, -1, 3, 2};
int c[2][2] = {2, 1, 5, 10};
int d[2][2] = {1, -4, 2, -2};
int result1[2][2], result2[2][2];

int main(){
    calculate();

    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            printf("%d ", result1[i][j]);
        }
        printf("\n");
    }
    for(int i = 0; i < 2; i++) {
        for(int j = 0; j < 2; j++) {
            printf("%d ", result2[i][j]);
        }
        printf("\n");
    }
}

void calculate(void){
    result1[0][0] = a[0][0] * b[0][0] + a[0][1] * b[1][0];
    result1[0][1] = a[0][0] * b[0][1] + a[0][1] * b[1][1];
    result1[1][0] = a[1][0] * b[0][0] + a[1][1] * b[1][0];
    result1[1][1] = a[1][0] * b[0][1] + a[1][1] * b[1][1];

    result2[0][0] = c[0][0] * d[0][0] + c[0][1] * d[1][0];
    result2[0][1] = c[0][0] * d[0][1] + c[0][1] * d[1][1];
    result2[1][0] = c[1][0] * d[0][0] + c[1][1] * d[1][0];
    result2[1][1] = c[1][0] * d[0][1] + c[1][1] * d[1][1];
}
```

4. Results and discussions
   Console output:

```
5 -5
11 -6
4 -10
25 -40
```

   The program outputs the correct results as expected.