1. انجام دهید!

- با استفاده از حلقه for برنامه ای بنویسید که عدد صحیح x را از کاربر گرفته و حاصل $x!$ را محاسبه کند و در خروجی چاپ کند.

- برنامه ای که در قسمت قبل نوشتید را به صورت یک تابع به نام fact دربیاورید. این تابع عدد صحیح x را به عنوان آرگومان[1] گرفته، سپس حاصل $x!$ را حساب کرده و به عنوان خروجی بازمی‌گرداند.

- برنامه ای در تابع main بنویسید که عدد n را از کاربر بگیرد و مقدار صحیح زیر را با استفاده از تابع fact حساب کرده و در خروجی چاپ کند:

$$\sum_{i=0}^{n}(-1)^i(i!)$$

- اکنون اظهار[2] تابع fact را در یک header file به نام fact.h قرار دهید و بدنه ی آن را درون یک فایل cpp با نام fact.cpp قرار دهید. دقت کنید که fact.h را در ابتدای fact.cpp، include کرده باشید. محتوای دو فایل fact.h و fact.cpp در زیر آمده است:

```
fact.h
int fact(int x);
```

```
fact.cpp
#include "fact.h"
int fact(int x) {
    /* Your Code Goes here. */
}
```

حال با توضیحات بالا قسمت قبل را دوباره انجام دهید:

```
main.cpp
#include "fact.h"
int main() {
    /* Your Code Goes here. */
}
```

[1] argument
[2] declaration

قسمت 1 : موارد خواسته شده را انجام داده و نتایج به دست آمده را در کادر زیر بنویسید.

```c
#include <stdio.h>

int main(){
	int x = 0;
	int result = 1;
	printf("Please Enter x: ");
	scanf("%d", &x);
	for (int i = 1; i <= x; i++){
		result *= i;
	}
	printf("%d! = %d\n", x, result);
	return 0;
}
```

```c
int fact(int x){
	int result = 1;
	for (int i = 1; i <= x; i++){
		result *= i;
	}
	return result;
}
```

```c
int main(){
	int n , result = 0;
	printf("Please Enter n: ");
	scanf("%d", &n);
	for (int i = 0; i <= n; i++){
		if (i % 2 == 0){
			result += fact(i);
		}
		else{
			result -= fact(i);
		}
	}
	printf("The result is: %d\n", result);
	return 0;
}
```

اظهار تابع fact در فایل fact.h قرار می گیرد و بدنه آن که در بالا تعریف شده به همان شکل در فایل fact.c قرار می گیرد و main.c به این شکل در می آید:

```c
#include <stdio.h>
#include "fact.h"
int main(){
	int n, result = 0;
	printf("Please Enter n: ");
	scanf("%d", &n);
	for (int i = 0; i <= n; i++){
		if (i % 2 == 0){
			result += fact(i);
		}
		else{
			result -= fact(i);
		}
	}
	printf("The result is: %d\n", result);
	return 0;
}
```

2. انجام دهید!

- در این قسمت قرار است تا تابع فیبوناچی را حساب کنید. یادآوری می شود که:

$$F(n) = F(n-1) + F(n-2);$$

$$F(1) = F(2) = 1;$$

انتظار می‌رود که تابع شما به عنوان ورودی یک عدد long گرفته و همچنین یک عدد long را به عنوان مقدار بازگشتی بدهد.

```
Long    fibonacci (long n) {
        // your code goes here
}
```

- تابع پیاده‌سازی شده در بالا را به گونه ای تغییر دهید تا در صورتی که کاربر عدد نامناسبی را وارد نمود (برای مثال اعداد منفی) مقدار 0 را به عنوان خطا بازگرداند.

قسمت 2 : : موارد خواسته شده را انجام داده و نتایج به دست آمده را در کادر زیر بنویسید.

با ورود عدد نامناسب توسط کاربر در این کد ورودی منفی صفر برگردانده می شود. حال به خاطر این که اعداد اعشاری هم ورودی نامناسب حساب می شوند برای این که ورودی اعشاری هم صفر برگردانده بشود باید ورودی تابع به صورت اعشاری گرفته بشود تا بتوان در تابع آن را بررسی کرد و در صورت اعشاری بودن صفر برگرداند پس تعریف تابع را نیز باید عوض کرده و تغییر کامنت شده را در کد ایجاد می کنیم.

```
long fibonacci(long n){
	long first_num = 1, second_num = 1 , new_num = 0;
	if (n <= 0){
		return 0;
	}
// For this part the input of function must be in float or double
// This must be the function --> long fibonacci(float n);
//	if (n != (int)n){
//		return 0;
//	}
	else if (n == 1 || n == 2){
		return 1;
	}
	for (long i = 0; i < n - 2; i++){
		new_num = first_num + second_num;
		first_num = second_num;
		second_num = new_num;
	}
	return new_num;
}
```

3. فکر کنید!

1. به قطعه کد زیر نگاه کنید. به نظر شما بعد از اجرای برنامه چه اتفاقی می‌افتد؟

```
long fib(long n)
{
    if (n <= 2)
        return 1;
    return fib(n - 1) + fib(n - 2);
}
```

قسمت 3 : نتیجه را در کادر زیر توضیح دهید.

این تابع راه حلی بازگشتی برای به دست آوردن مقادیر تابع Fibonacci است یعنی راه حلی برای بخش قبل است. در این تابع هر بار به ازای ورودی n دو مقدار n-1 و n-2 به تابع داده می شوند حال دوباره تابع را با مقادیرn-1 و n-2 صدا زده و انجام می دهیم یعنی این بار مقادیر n-2 وn-3 ورودی جدید تابع در n-1 و مقادیر n-3 و n-4 ورودی جدید تابع در n-2 هستند این کار به همین شکل ادامه پیدا می کند تا ورودی کم تر یا مساوی 2 شود که در این صورت مقدار 1 برگردانده می شود حال برمی گردیم و هر بار به همین ترتیب به قبل (عقب) رفته و مقادیر جدید را برمی گردانیم تا در نهایت مقدار اصلی که مقدار تابع Fibonacci به ازای ورودی n است به عنوان خروجی برگردانده شود.

در واقع همان رابطه بازگشتی Fibonacci در این کد آورده شده است.

4. انجام دهید!

1. یک فایل جدید ایجاد کرده و برنامه زیر را در آن بنویسید و سعی کنید آن را کامپایل نمایید.

```c
#include <stdio.h>

int main() {

	int z = 4;
	if (z - 4) {
		int I = 1;
		z = z + I;
	}
	else {
		int I = 1;
		z = I + 1;
		{
			int I = 1;
			z = I + 1;
		}
		z = I;
	}
	z = z + I;
	do {
		int I = 0;
		z = I + 1;
		i++;
	} while (I < 15);

	return 0;

}
```

قسمت 4 :

1) چرا این برنامه کامپایل نمی شود؟ در کادر زیر توضیح دهید.
2) کلیه ی خطاهای کامپایلی این کد را **با ذکر شماره خط** بیان کرده و آن ها را رفع کنید. نتایج به دست آمده را در کادر زیر بنویسید.

اولین خطا در خط 17 ایجاد می شود و دلیل آن این است که I در فضای main تعریف نشده است و تمامی I هایی که قبل از این خط تعریف شده اند در فضای داخل تری هستند و I این خط به آن ها دسترسی ندارد برای رفع این مشکل می توان در فضای main متغیر I را تعریف و مقدار دهی کرد.

دومین خطا در خط 21 ایجاد می شود که دلیل آن این است که اصلا عبارتی با نام i در هیچ جایی تعریف نشده است برای حل این خطا می توان i را به I تبدیل کرد تا هم تعریف شده باشد و هم حلقه که با مقدار I انجام می شود به مشکل نخورد.

سومین خطا در خط 22 ایجاد می شود که دلیل آن این است که حلقه while که با مقدار I کار می کند در فضای main تعریف شده است ولی متغیر I در فضای main تعریف نشده است برای همین برنامه این متغیر را نمی شناسد برای حل این مشکل مثل مشکل اول می توان I را در فضای main تعریف کرد (به عبارتی خارج از if و else و داخل main) و مقدار داد تا مشکل حل شود با این کار خطای دوم نیز به همان شکل گفته شده درست می شود.

پس برای این که خطا های کامپایل برطرف شوند باید I در فضای main تعریف شود و i در خط 21 به I تبدیل شود (همچنین برای آن که در لوپ بی نهایت قرار نگیریم خط 19 را باید حذف کنیم).

5. انجام دهید!

قسمت 5: برنامه زیر را اجرا کنید و سپس مقدار خروجی برنامه را نوشته و هر مقدار چاپ شده را توجیه کنید. این نتایج را در کادر تعبیه شده بنویسید.

```
int g(int y){
      int j = 2;
      y = j * 3;
      return y;
}

int f(int x){
      return g(x);
}

int main()
{
      int a = 10;
      printf("%d", f(a));
      return 0;
}
```

در ابتدا مقدار 10 به تابع f داده می شود در تابع f تابع g با مقدار x که برابر 10 است صدا زده می شود حال به تابع g می رویم در این تابع مقدار y که ورودی تابع است در ابتدا برابر 10 خواهد بود سپس مقدار y برابر ضرب مقدار j که برابر 3 است در 2 می شود پس y مقدار جدید 6 را گرفته و برگردانده می شود حال مقدار 6 به تابع f می رود و در آن جا نیز بدون تغییری برگردانده می شود و در نهایت مقدار 6 به main برمی گردد و توسط printf چاپ می شود پس خروجی این برنامه برابر 6 خواهد بود و اصلا به مقدار a وابسته نخواهد بود.

6 . انجام دهید! (امتیازی)

۱) خروجی این برنامه های زیر چیست ؟ دلیل خود را به صورت کامل تشریح کنید .

```c
#include<stdio.h>
int function();
main()
{
    int i;
    i = function();
    printf("%d", i);
    return 0;
}
function()
{
    int a;
    a = 250;
}
```

```c
#include<stdio.h>

void dynamic(int s, ...)
{
    printf("%d ", s);
}

int main()
{
    dynamic(2, 4, 6, 8);
    dynamic(3, 6, 9);
    return 0;
}
```

```c
#include<stdio.h>
void fun(int x)
{
  if(x > 0)
  {
      fun(--x);
      printf("%d\t", x);
      fun(--x);
  }
}
int main()
{
  int a = 4;
  fun(a);
  getchar();
  return 0;
}
```

کد بالا سمت چپ: در این کد چون تابع function هیچ مقداری را با این که خروجی آن باید int باشد برنمی گرداند مقداری نامعلوم return می شود پس مقدار i برابر مقداری نامعلوم می شود و عددی غیرمعمول چاپ خواهد شد و همچنین چون i مقدار دهی شده است حتی با این که این مقدار تعریف شده نیست و نامعمول است به خطای runtime نمی خوریم (به دلیل وجود int در تعریف تابع function به عنوان خروجی حتما باید مقداری int برگردانده شود چون ما مقداری را در کدمان به عنوان خروجی تابع مشخص نکرده ایم مقداری نامعلوم که عدد پرتی است return خواهد شد)

کد بالا سمت راست: دراین کد تابع dynamic به خاطر 3 نقطه ای که در ورودی آن آمده است می تواند هر تعدادی ورودی دریافت کند حال اولین ورودی که به تابع داده می شود در متغیر s ذخیره می شود یعنی در کد، خط اول main مقدار 2 را چاپ می کند و خط بعدی مقدار 3 را چاپ می کند و استفاده ای از مقادیر ورودی دیگر نمی شود پس خروجی به شکل 2 یک فاصله و 3 خواهد بود.(3 2)

کد پایین سمت چپ: ابتدا مقدار 4 به تابع fun داده می شود. این تابع در fun(--x) اول هر بار می رود تا در نهایت به مقدار x صفر می رسد، در برگشت مقدار x برابر 1-1 یعنی صفر می شود این صفر چاپ شده و در fun(--x) بعدی با مقدار -1 می رود که در آن به خاطر شرط موجود در تابع اتفاقی نمی افتد حال به x برابر 2 برمی گردیم در آن جا 1-2 چاپ می شود و تابع با مقدار 0 صدا زده می شود که در آن به خاطر شرط موجود در تابع اتفاقی نمی افتد حال به x برابر 3 برمی گردیم مقدار 1-3 چاپ شده و تابع با مقدار 1 دوباره صدا زده می شود این بار در تابع 0 چاپ می شود چون تابع درست مثل زمانی که x برابر 1 بود عمل می کند حال به x برابر 4 برمی گردیم مقدار 1-4 چاپ شده و تابع با مقدار 2 دوباره صدا زده می شود و به همان شکلی که x برابر 2 بود عمل می کند و 0 و 1 چاپ می شود سپس تابع به main برمی گردد و منتظر دریافت یک کاراکتر از کاربر می ماند و با دریافت کاراکتر برنامه به پایان می رسد. به خاطر وجود \t بین تمام اعداد چاپی یک tab فاصله وجود خواهد داشت. پس خروجی برنامه در نهایت به شکل زیر خواهد بود:

0 1 2 0 3 0 1

موفق باشید