Examen 01 - Programación e introducción a los métodos numéricos

William Oquendo, woquendo@gmail.com

1 Suma de inverso de cuadrados

(1.0/5.0) Adapte el archivo `suminversesquares.cpp` para que, usando un bucle, calcule la suma de los inversos de los cuadrados de los números naturales menores a un número dado que se leerá como argumento en la línea de comandos. Por ejemplo, para el número 4 le corresponde la suma $1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 \simeq$ `1.423611111111111e+00`. El programa debe imprimir a la pantalla dos columnas en donde la primera columna tenga los número de 1 hasta el que se dió como argumento y en la segunda columna la suma de los cuadrados menores o iguales de cada uno de esos números. Ejemplo: Asumiendo que compila como

```
g++ suminversesquares.cpp
```

al ejecutar

```
./a.out 500
```

debe obtener

1	1.000000000000000e+00
2	1.250000000000000e+00
3	1.361111111111111e+00
⋮	⋮
498	1.642928049466766e+00
499	1.642932065514894e+00
500	1.642936065514894e+00

Utilice el tipo apropiado de datos para la suma y para el contador. **Imprima los valores de la suma en notación exponencial y con 15 cifras decimales de precisión**.

El archivo ya se encuentra en el repositorio y se transcribe a continuación

```cpp
#include <iostream>
#include <cstdlib>

void printsum(int nmax);

int main(int argc, char *argv[])
{
  const int NMAX = std::atoi(argv[1]);

  printsum(NMAX);

  return 0;
}

void printsum(int nmax)
{
  fill here and remove this line
}
```

2 N-ésimo número primo

(2.0/5.0) Los primeros seis números primos son $2, 3, 5, 7, 11, 13$. Por tanto, el cuarto número primo es 7. Cuál es el n-ésimo número primo? Para esto modifique el archivo `nthprime.cpp` para que calcule el n-ésimo número primo. Al ejecutar el programa se debe dar el argumento que indica cuál es el valor de n (para encontrar n-ésimo número primo). Para el n de ejemplo, el tiempo máximo de ejecución será de 8 segundos.

Ejemplo de ejecución:

```
./a.out 99810
1296983
```

Plantilla base (ya se encuentra en el repositorio)

```
#include <iostream>
#include <cstdlib>
#include <cmath>

int isprime(int m);
int nthprime(int n);

int main(int argc, char *argv[])
{
  const int n = std::atoi(argv[1]);
  std::cout << nthprime(n) << "\n";

  return 0;
}

int isprime(int m)
{
    // Esta funcion retorna 1 si m es primo, 0 en otro caso
    // fill here
}

int nthprime(int n)
{
    // esta funcion retorna el n-esimo primo, haciendo uso de isprime
    // fill here
}
```

3 Campo eléctrico a partir de potencial eléctrico

(2.0/5.0) En una dimensión, el campo E_x y el potencial V eléctricos están relacionados por

$$E_x = -\frac{dV}{dx}.$$

Asuma que en la región $0 \leq x \leq 2.5$ el potencial está dado por

$$V(x) = 2x \sin x.$$

Calcule el campo en esta región usando extrapolación de Richardson aplicada a la derivada central, con $h = \Delta x$ que se lee de la línea de comandos. El programa debe escribir los datos a la pantalla en tres columnas: valor de x, valor exacto de la derivada, y valor numérico, todo impreso en notación científica con 16 decimales.

El programa debe llamarse `field.cpp` . La plantilla se encuentra en el repositorio y se muestra a continuación:

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using fptr = double(double);
double potential(double x);
double dev_central(double x, double h, fptr fun);
template <typename algptr>
double dev_richardson(double x, double h, fptr fun, algptr algo);


int main(int argc, char **argv)
{
  std::cout.precision(16); std::cout.setf(std::ios::scientific);

  const double h = std::atof(argv[1]);

  for(double x = 0.0; x <= 2.5; x += h) {
      // fill here
  }

  return 0;
}


double potential(double x)
{
```

```
  // fill here
}

double dev_central(double x, double h, fptr fun)
{
  // fille here
}

template <typename algptr>
double dev_richardson(double x, double h, fptr fun, algptr algo)
{
    // fill here
}
```

Como ejemplo, cuando `h=0.8` se obtiene

```
./a.out 0.8
0.0000000000000000e+00  -0.0000000000000000e+00 -0.0000000000000000e+00
8.0000000000000004e-01  -2.5494429167545105e+00 -2.5425494854297019e+00
1.6000000000000001e+00  -1.9057087347188859e+00 -1.8974820180509955e+00
2.4000000000000004e+00  2.1885634734956789e+00  2.1912121841183239e+00
```

y cuando `h=0.1`

```
./a.out 0.1
0.0000000000000000e+00  -0.0000000000000000e+00 -0.0000000000000000e+00
1.0000000000000001e-01  -3.9866766634926143e-01 -3.9866741700345171e-01
...
2.3000000000000007e+00  1.5734592735337549e+00  1.5734601881145676e+00
2.4000000000000008e+00  2.1885634734956811e+00  2.1885641429501157e+00
```