

**Benemérita Universidad Autónoma de Puebla**

**Facultad de Ciencias de la Computación**

**Ingeniería en Tecnologías de la Información**

**Ingeniería de software II**

**Adrian Jesus Lora Molina 201404523**

**Román Huerta Bernabé 201400473**

**Juan Betanzo Lucero 201426711**

**Programa: Integrador**

Escribe un programa para realizar una integración numérica.

**Requisitos dados**

Requisitos: escribir un programa para integrar numéricamente una función usando la regla de Simpson y escribir la función para la distribución normal. El programa debe estar diseñado para integrarse utilizando varias funciones proporcionadas ... Necesitará este programa para calcular los valores de las diversas distribuciones estadísticas utilizadas en asignaciones posteriores del programa y en el análisis de los datos de su PSP.

Prueba: prueba a fondo el programa. Incluya una prueba para calcular los valores de probabilidad de la integral de distribución normal desde -infinito hasta x = 2,5, desde -infinito hasta x = 0,2 y desde -infinito hasta x = -1,1. Los resultados deberían ser aproximadamente 0,9938, 0,5793 y 0,1357 respectivamente. Incluya en su informe de prueba una tabla de resultados en el formato de la Tabla D10.

**Table 1. Test Results Format**

|  |  |  |
| --- | --- | --- |
| Test Value -- X | Expected Results | Actual Results |
| 2.5 | 0.9938 |  |
| 0.2 | 0.5793 |  |
| -1.1 | 0.1357 |  |

**Diseño**

Para encapsular el concepto de una función de una sola variable, crearé una clase base muy simple, single\_variable\_function, con una característica: at (x: double). Llamar a con un valor devolverá el valor de la ecuación, evaluado en el valor dado. Esto se utilizará como base para la clase normal\_distribution\_base (la función que se integra para obtener nuestro valor deseado) y la clase normal\_distribution\_from\_negative\_infinity (nombre largo, pero descriptivo; su función at devuelve el valor de la integral normal evaluada de infinito negativo a el valor dado, manejando la "solución infinita" descrita en la sección de planificación).

La mayor parte de ese trabajo lo realizará la clase simpson\_integrator, que encapsula la lógica de integración de Simpson y puede integrar cualquier función de variable única que le echemos.

Para manejar la E / S, nuestro valiente analizador de entrada simple se reutiliza una vez más, para formar el analizador de entrada normal\_distribution\_input\_parser, que toma una serie de dobles de la entrada estándar e imprime los valores resultantes en la salida estándar.

**Code**

**main.cpp**

|  |
| --- |
| /\*  \*/  #include <fstream>  #include <iostream>  #include "string.h"  #ifndef SINGLE\_FUNCTION\_INPUT\_PARSER\_H  #include "single\_function\_input\_parser.h"  #endif  #ifndef NORMAL\_DISTRIBUTION\_INTEGRAL\_H  #include "normal\_distribution\_integral.h"  #endif  istream \*  input\_stream\_from\_args( int arg\_count, const char\*\* arg\_vector )  {  istream\* Result = NULL;  if ( arg\_count == 1 )  {  Result = &cin;  }  else  {  const char\* help\_text = "PSP exercise 5A: Integrate the normal distribution from\nnegative infinity to the values given from standard input\nUsage: \n\tpsp\_5a\n\n";  cout << help\_text;  }  return Result;  }  int main( int arg\_count, const char\*\* arg\_vector )  {  //get the input stream, or print the help text as appropriate  istream\* input\_stream = input\_stream\_from\_args( arg\_count, arg\_vector );  if ( input\_stream != NULL )  {  single\_function\_input\_parser parser;  parser.set\_input\_stream( input\_stream );  normal\_distribution\_integral f;  parser.set\_function( &f );  parser.parse\_until\_eof();  }  }  /\*  \*/ |

**single\_variable\_function**

|  |
| --- |
| /\*  \*/  #ifndef SINGLE\_VARIABLE\_FUNCTION\_H  #define SINGLE\_VARIABLE\_FUNCTION\_H  class single\_variable\_function  {  public:  //the value of the function, evaluated at x  virtual double at( double x ) const = 0;  };  #endif  /\*  \*/ |

**normal\_function\_base**

|  |
| --- |
| /\*  \*/  #ifndef NORMAL\_FUNCTION\_BASE\_H  #define NORMAL\_FUNCTION\_BASE\_H  #ifndef SINGLE\_VARIABLE\_FUNCTION\_H  #include "single\_variable\_function.h"  #endif  class normal\_function\_base : public single\_variable\_function  {  public:  virtual double at( double x ) const;  };  #endif  /\*  \*/ |

|  |
| --- |
| /\*  \*/  #include "normal\_function\_base.h"  #include <math.h>  double  normal\_function\_base::at( double x ) const  {  const double base\_value = 1 / ( sqrt( M\_PI \* 2 ) );  return base\_value \* exp( -(x \* x ) / 2 );  }  /\*  \*/ |

**normal\_distribution\_integral**

|  |
| --- |
| #ifndef NORMAL\_DISTRIBUTION\_INTEGRAL\_H  #define NORMAL\_DISTRIBUTION\_INTEGRAL\_H  #ifndef NORMAL\_FUNCTION\_BASE\_H  #include "normal\_function\_base.h"  #endif  #ifndef SIMPSON\_INTEGRATOR\_H  #include "simpson\_integrator.h"  #endif  class normal\_distribution\_integral : public single\_variable\_function  {  public:  virtual double at( double x ) const;  protected:  simpson\_integrator homer;  normal\_function\_base base;  };  #endif |

|  |
| --- |
| #include "normal\_distribution\_integral.h"  #ifndef CONTRACT\_H  #include "contract.h"  #endif  double  normal\_distribution\_integral::at( double x ) const  {  double Result = 0;  if ( x < 0 )  {  Result = 0.5 - homer.integral( base, 0, 0-x );  }  else if ( x > 0 )  {  Result = 0.5 + homer.integral( base, 0, x );  }  else if ( x == 0 )  {  Result = 0.5;  }  else  {  CHECK( false );  }  return Result;  } |

**simpson\_integrator**

|  |
| --- |
| /\*  \*/  #ifndef SIMPSON\_INTEGRATOR\_H  #define SIMPSON\_INTEGRATOR\_H  #ifndef SINGLE\_VARIABLE\_FUNCTION\_H  #include "single\_variable\_function.h"  #endif  class simpson\_integrator  {  public:  //sets the acceptable error in the computation  void set\_acceptable\_error( double new\_error );  //sets the starting number of segments  void set\_starting\_segment\_count( int new\_starting\_segment\_count );  //returns the simpson integral of the given single variable function between the two limits  double integral( const single\_variable\_function& f, double lower\_limit, double upper\_limit ) const;  //constructor  simpson\_integrator::simpson\_integrator( void );  protected:  double acceptable\_error;  int starting\_segment\_count;  double single\_pass( const single\_variable\_function& f, double lower\_limit, double upper\_limit, int segment\_count ) const;  bool is\_odd( int x ) const;  };  #endif  /\*  \*/ |

|  |
| --- |
| /\*  \*/  #include "simpson\_integrator.h"  #ifndef CONTRACT\_H  #include "contract.h"  #endif  void  simpson\_integrator::set\_acceptable\_error( double new\_error )  {  REQUIRE( new\_error > 0 );  acceptable\_error = new\_error;  }  void  simpson\_integrator::set\_starting\_segment\_count( int new\_starting\_segment\_count )  {  REQUIRE( new\_starting\_segment\_count > 0 );  REQUIRE( !is\_odd( new\_starting\_segment\_count ) );  starting\_segment\_count = new\_starting\_segment\_count;  }  simpson\_integrator::simpson\_integrator( void ) :  acceptable\_error( 0.00001 ),  starting\_segment\_count( 20 )  {  }  double  simpson\_integrator::integral( const single\_variable\_function& f,  double lower\_limit, double upper\_limit ) const  {  double Result = 0;  double previous\_result = 0;  double last\_error = 0;  int segment\_count = starting\_segment\_count;  bool result\_has\_been\_calculated = false;  while ( !result\_has\_been\_calculated  || ( last\_error > acceptable\_error ) )  {  previous\_result = Result;  Result = single\_pass( f, lower\_limit, upper\_limit, segment\_count );  last\_error = fabs( Result - previous\_result );  //cout << "Segments: " << segment\_count << "; error: " << last\_error << "\n";  segment\_count \*= 2;  result\_has\_been\_calculated = true;  }  return Result;  }  double  simpson\_integrator::single\_pass( const single\_variable\_function& f,  double lower\_limit, double upper\_limit,  int segment\_count ) const  {  double Result = 0;  const double segment\_width = ( upper\_limit - lower\_limit ) / static\_cast<double>(segment\_count);  const double third\_width = segment\_width / static\_cast< double >(3);  Result = f.at( lower\_limit ) \* third\_width;  //cout << "From " << lower\_limit << " to " << upper\_limit << "\n";  //cout << "Width " << segment\_width << "\n";  //cout << "Term 0: " << Result << "\n";  for ( int i = 1; i < segment\_count; ++i )  {  const double xi = lower\_limit + static\_cast< double >( i ) \* segment\_width;  const double segment\_value = f.at( xi ) \* third\_width;  //cout << "Xi: " << xi << " ";  //cout << "F: " << f.at( lower\_limit + static\_cast<double>(i) \* segment\_width ) << " ";  if ( is\_odd(i) )  {  Result += 4 \* segment\_value;  //cout << "Term " << i << ": " << 4 \* segment\_value << "\n";  }  else  {  Result += 2 \* segment\_value;  //cout << "Term " << i << ": " << 2 \* segment\_value << "\n";  }  }  //cout << "f( upper ): " << f.at( upper\_limit ) << "\n";  Result += f.at( upper\_limit ) \* third\_width;  CHECK( fabs( upper\_limit - ( lower\_limit + segment\_count \* segment\_width ) ) < acceptable\_error );  return Result;  }  bool  simpson\_integrator::is\_odd( int x ) const  {  return ( x & 1 );  }  /\*  \*/ |

**single\_function\_input\_parser**

|  |
| --- |
| /\*  \*/  #ifndef SINGLE\_FUNCTION\_INPUT\_PARSER\_H  #define SINGLE\_FUNCTION\_INPUT\_PARSER\_H  #ifndef SIMPLE\_INPUT\_PARSER\_H  #include "simple\_input\_parser.h"  #endif  #ifndef SINGLE\_VARIABLE\_FUNCTION\_H  #include "single\_variable\_function.h"  #endif  class single\_function\_input\_parser : public simple\_input\_parser  {  public:  virtual void parse\_last\_line( void );  void set\_function( single\_variable\_function\* const new\_function );  single\_function\_input\_parser( void );  protected:  single\_variable\_function\* f;  };  #endif  /\*  \*/ |

|  |
| --- |
| /\*  \*/  #include "single\_function\_input\_parser.h"  #ifndef CONTRACT\_H  #include "contract.h"  #endif  void  single\_function\_input\_parser::parse\_last\_line( void )  {  REQUIRE( f != NULL );  char\* conversion\_end = NULL;  double new\_x = strtod( last\_line().c\_str(), &conversion\_end );  if ( conversion\_end == last\_line().data() )  {  cerr << "Not a double : " << last\_line() << "\n";  }  else  {  cout << new\_x << ": " << f->at( new\_x ) << "\n";  }  }  void  single\_function\_input\_parser::set\_function( single\_variable\_function\* const new\_function )  {  f = new\_function;  }  single\_function\_input\_parser::single\_function\_input\_parser( void ):  f( NULL )  {  }  /\*  \*/ |

**Table 3. Test Results Format**

|  |  |  |
| --- | --- | --- |
| Test Value -- X | Expected Results | Actual Results |
| 2.5 | 0.9938 | 0.99379 |
| 0.2 | 0.5793 | 0.57926 |
| -1.1 | 0.1357 | 0.135666 |

**PSP1.1 Project Plan Summary**

**Table 4. Project Plan Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Adrian Jesus Lora Molina | Date: | 210621 |
| Program: | Integrador numérico | Program# | Preoyecto |
| Instructor: | Juan Manuel Gonzalez Calleros | Language: | C++ |

|  |  |  |  |
| --- | --- | --- | --- |
| Summary | Plan | Actual | To date |
| Loc/Hour | 50 | 34 | 48 |
| Planned time | 186 |  | 186 |
| Actual time |  | 182 | 182 |
| CPI (cost/performance index) |  |  | 1.02 |
| %reused | 22 | 31 | 31 |

|  |  |  |  |
| --- | --- | --- | --- |
| Program Size | Plan | Actual | To date |
| Base | 25 | 25 |  |
| Deleted | 0 | 5 |  |
| Modified | 5 | 1 |  |
| Added | 155 | 103 |  |
| Reused | 52 | 55 | 142 |
| Total New and Changed | 160 | 104 | 790 |
| Total LOC | 232 | 178 | 1144 |
| Total new/reused | 0 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time in Phase (min): | Plan | Actual | To Date | To Date% |
| Planning | 26 | 67 | 183 | 18 |
| Design | 19 | 24 | 103 | 10 |
| Code | 50 | 42 | 268 | 27 |
| Compile | 11 | 13 | 65 | 6 |
| Test | 65 | 25 | 312 | 31 |
| Postmortem | 15 | 11 | 77 | 8 |
| Total | 186 | 182 | 1008 | 100 |
| *Defects Injected* |  | Actual | To Date | To Date % |
| Plan |  | 0 | 0 | 0 |
| Design |  | 4 | 25 | 25 |
| Code |  | 15 | 70 | 69 |
| Compile |  | 0 | 3 | 3 |
| Test |  | 0 | 3 | 3 |
| Total development |  | 19 | 101 | 100 |
| *Defects Removed* |  | Actual | To Date | To Date % |
| Planning |  | 0 | 0 | 0 |
| Design |  | 0 | 0 | 0 |
| Code |  | 3 | 22 | 21 |
| Compile |  | 14 | 49 | 49 |
| Test |  | 2 | 30 | 30 |
| Total development |  | 19 | 101 | 100 |
| After Development |  | 0 | 0 |  |

**Time Recording Log**

**Table 5. Time Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Roman Huerta Bernabe | Date: | 210621 |
|  |  | Program: | Integrador numérico |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start | Stop | Interruption Time | Delta time | Phase | Comments |
| 210621 11:29:59 | 210621 12:58:05 | 21 | 67 | plan |  |
| 210621 13:10:06 | 210621 13:34:09 | 0 | 24 | design |  |
| 210621 13:54:41 | 210621 14:37:21 | 0 | 42 | code |  |
| 210621 14:39:39 | 210621 14:53:06 | 0 | 13 | compile |  |
| 210621 14:53:10 | 210621 15:18:55 | 0 | 25 | test |  |
| 210621 16:42:05 | 210621 16:53:14 | 0 | 11 | postmortem |  |
|  |  |  |  |  |  |

**Defect Reporting Logs**

**Table 6. Defect Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Juan Huerta Bernabe | Date: | 210621 |
|  |  | Program: | Integrador numérico |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect found | Type | Reason | Phase Injected | Phase Removed | Fix time | Comments |
| 210621 14:08:47 | md | ig | design | code | 9 | característica separada agregada para single\_pass del procedimiento de simpson |
| 210621 14:18:26 | md | ig | design | code | 0 | característica is\_odd agregada |
| 210621 14:29:02 | md | ig | design | code | 1 | no diseñó para manejar diferentes funciones. Probablemente innecesario, pero es más modular |
| 210621 14:39:50 | sy | om | code | compile | 0 | olvidé cambiar la inclusión del encabezado |
| 210621 14:40:31 | sy | ty | code | compile | 0 | Error tipográfico en constante de cadena |
| 210621 14:40:58 | sy | om | code | compile | 0 | Olvidé agregar const a la implementación |
| 210621 14:42:04 | iu | cm | code | compile | 0 | Orden incorrecto de argumentos |
| 210621 14:43:25 | sy | ty | code | compile | 0 | olvidé agregar const a la implementación |
| 210621 14:43:50 | wn | cm | code | compile | 0 | nombre incorrecto de una variable |
| 210621 14:44:49 | wt | ig | code | compile | 0 | intentó "constificar" un miembro y luego modificarlo |
| 210621 14:45:48 | sy | ty | code | compile | 0 | olvidé agregar "single\_function\_input\_parser ::" al constructor |
| 210621 14:46:42 | md | om | design | compile | 0 | olvidé asignar una integral normal al analizador de entrada |
| 210621 14:49:02 | sy | ty | code | compile | 0 | Falta una coma en la lista de constructores |
| 210621 14:49:26 | sy | om | code | compile | 0 | se perdieron más consts en la implementación |
| 210621 14:50:10 | sy | ty | code | compile | 0 | intento de asignación al estilo eiffel (: =) en la función |
| 210621 14:50:44 | sy | ty | code | compile | 0 | nombre de variable mal escrito |
| 210621 14:51:22 | wt | cm | code | compile | 1 | usó "doble" como argumento en is\_odd; destinado a usar int |
| 210621 14:54:09 | wa | ig | code | test | 18 | el cheque fue por doble igualdad; debería haber usado diferencia <margen |
| 210621 15:13:26 | wa | om | code | test | 3 | cierta confusión en las condiciones de verificación booleana para continuar el proceso de simpson |
|  |  |  |  |  |  |  |