POLTAVA UNIVERSITY OF ECONOMICS AND TRADE

EDUCATIONAL AND SCIENTIFIC INSTITUTE OF INTERNATIONAL EDUCATION

FORM OF DAY EDUCATION
DEPARTMENT OF COMPUTER SCIENCES AND INFORMATION TECHNOLOGY

Allowed for protection
Head of the department _____ Olkhovska O.V.
(signature)
"_____"_____2023

EXPLANATORY NOTE FOR THE GRADUATE THESIS

«DEVELOPMENT OF SIMULATOR SOFTWARE ON THE TOPIC "DERIVATIVES" OF THE DISTANCE LEARNING COURSE "HIGHER AND APPLIED MATHEMATICS»

from specialty 122 «Computer science»
educational program «Computer science»
bachelor's degree

The executor of the work is Samuel AMPAI
_____ "___"_____2023.
(signature)
Scientific supervisor, Ph.D. k.ped.s., Oksana KOSHOVA

_____ "___"_____2023.
(signature)

Poltava 2023

POLTAVA UNIVERSITY OF ECONOMICS AND TRADE

I APPROVE

Head of the department

_____Olkhovska O.V.

" ___» _____2023

TASKS AND CALENDAR SCHEDULE
OF EXECUTION OF DIPLOMA THESIS
Graduate of higher education in specialty 122 "Computer science"
Educational program "Computer Science"
Surname, first name, patronymic _ Samuel Ampai _
1. **The topic «Development of simulator software on the topic "Derivatives" of the distance learning course "Higher and applied mathematics»**
approved by the rector's order No. ____-H from___2022.
The deadline for the student to submit a thesis «__» _____ 2023.
2. Source data for the bachelor's thesis: publications on the topic of educational simulators in distance courses in computer science.
3. Content of the explanatory note (list of issues to be developed)
LIST OD SYMBOLS, UNITS, ABBREVIATIONS, RERMS
AN INTRODUCTION
CHAPTER 1. PROBLEM STATEMENT
1.1. Problem Statement
CHAPTER2. THEORETICAL PART
2.1. Derivative statement
2.2. The purposes of derivative
2.3. Basic derivative function
CHAPTER 3. PRACTICAL PART
3.1. Online GDB for C++
3.2. Components of Online GDB
3.3. The application of derivative in C++
3.4. Design And Programming of The Scientific Calculator
CONCLUSIONS
REFERENCES
APPENDIX A. Algorithm translation
Descriptive Algorithm
APPENDIX B. program code

4. List of graphic material: 3-4 sheets of block diagrams, other necessary illustrations.

5. Consultants of sections of the bachelor's thesis

Section	Surname, initials, position of consultant	Signature, date	
		issued the task	accepted the task
1. Statement of the problem	Oksana Koshova		
2. Information review	Oksana Koshova		
3. Theoretical part	Oksana Koshova		
4. Practical implementation	Oksana Koshova		

6. Calendar schedule of bachelor's work

The content of the work	Deadline	Actual performance
1. Introduction		
2. Study of methodological recommendations and standards and report to the manager		
3. Formulation of the problem		
4. Information review of library and Internet sources		
5. Theoretical part		
6. The practical part		
7. Completion of registration		
8. Student report at the department		
9. Editing (if necessary), reviewing		

Issue date of task " ___ " _____ 2023.
Ampai Samuel, a higher education graduate
Scientific supervisor, Ph.D., k.ped.n., Oksana Koshova.

The results of the thesis defense

The thesis was evaluated at _____
(Points, assessment according to the national scale, assessment according to ECTS)

Minutes of the meeting of the EC No. ____ of "____" _____ 2023.
Secretary of the EC_____ _____
(signature) (initials and surname)

I approve

Chief department _____

Ph.D.. k.ph.-m.s. O. Olkhovska

"____" _____ 2023

Agreed

Supervisor _____

associate professor, Ph.D. O. Koshova

"___" _____ 2023

Plan

Thesis of a higher education graduate with a bachelor's degree
majors 122 Computer science
educational program 122 Computer science
Surname, first name, patronymic Ampai Samuel
On the topic «Development of simulator software on the topic "Derivatives" of the distance learning course "Higher and applied mathematics»

Graduate of higher education _____ Samuel Ampai
"____" _____ 2023

ABSTRACT

The purpose of the Grade work. The purpose of the grade work is development of software for application of derivatives in C++ programming language.

The object of the Grade work is distance learning system for students.

The subject of the Grade work is software for application of derivatives in C++ programming language.

CONTENT

LIST OF SYMBOLS, UNITS, ABBREVIATIONS, TERMS

SYMBOLS, UNITS, ABBREVIATIONS, TERMS	Explanation of symbols, units, abbreviations, terms
Simulator	a program enabling a computer to execute program
F(x)	This symbol represents the function whose derivative we are attempting to compute.
D(X)	This symbol signifies a change in the value of x that is infinitesimal. It is used to define the derivative, which is defined as the limit of the change in f(x) divided by the change in x as dx approaches zero.
∇f(x)	This symbol represents the gradient of f(x), which is a vector that points in the direction of the greatest increase in f(x). It is often used in multivariable calculus to find the direction of steepest ascent
C++	Object-oriented programming, generic programming, templates, and exception handling are all characteristics of C++. It is a popular choice for designing large-scale applications, and many software developers and businesses worldwide utilize it.

INTRODUCTION

Topicality. Every generation believes they are experiencing the most exciting period of ongoing social and technological growth. However, since the invention of computers and the internet, the present appears to be a period of unparalleled advancement.

A formal language used to convey instructions to a computer is called a programming language. It is employed in the creation of websites, applications, and software. Although programming languages are made to be simple for people to read and write, they are also organized in a way that makes them simple for computers to understand. Programming languages come in a wide variety, each with an own syntax and set of coding guidelines. Java, Python, C++, and JavaScript are a few of the more well-liked programming languages.

In this work grade, I created a simulator based on the topic "Derivatives" in the C++ programming language.

Derivatives are used in C++ to solve optimization problems, represent physical phenomena, and evaluate the behavior of complicated systems.it helps in machine as well.

C++ lacks a built-in derivative operator, although it does include libraries and frameworks for dealing with derivatives.

These libraries enable you to execute numerical calculations to approximate a function's derivative at a particular point. The Boost C++ Libraries, which includes several tools for conducting numerical calculations, including differentiation, is a popular library for working with derivatives in C++.

The Armadillo C++ Library is another popular library that offers a variety of linear algebra and numerical analysis tools, including support for differentiation and integration.

C++ libraries for working with derivatives frequently emphasize low-level access to hardware resources and performance optimization.

For the first task in this grade work, I will initially study arrays in Java, clarify their types and methods of declaring them, and explain how to enter the values of different types inside them, and I will explain their advantages and disadvantages.

For the second task in this grade work, I will design and write a program in the Java language that implements a scientific calculator, and I will use the array to store the values that the user will enter and also to store the values resulting from the calculations executed through the scientific calculator.

In the era of globalization, we used Calculator so many times. We use it to do the calculation in a short time. In this perspective, I have made a java Scientific Calculator software. By using this software, we can easily calculate our mathematical problem.

The purpose of the Grade work. The purpose of the grade work is development of software for application of derivatives in C++ programming language.

The object of the Grade work is distance learning system for students.

The subject of the Grade work is software for application of derivatives in C++ programming language.

The simulator’s goal is to be used as a training tool to help students learn the topic of derivatives in C++.

The following methods for the development have been used:

- NetBeans Platform including JDK.
- JFrame container.
- java language.
- java Swing.

C++ is a general-purpose programming language created by Bjarne Stroustrup at Bell Labs in the early 1980s. It is a C programming language extension that was developed to give extra capabilities for object-oriented programming (OOP), such as classes, inheritance, and polymorphism, while retaining the efficiency and low-level control of C.

The term "C++" is derived from the C increment operator, which is written as "++". In C++, the "++" operator reflects the language's ability to enhance C's capability.

C++ was first launched in 1985, and its combination of low-level control and high-level abstractions quickly garnered appeal among programmers. It was used to create a wide range of software, including operating systems, device drivers, and video games:

System specialized: C++ is not fundamentally platform-independent because code execution is dependent on the hardware that is underneath and operating system. C++ code, on the other hand, can be made platform-independent by employing platform-specific abstractions, such as using libraries that provide platform-independent functionality or writing code that works on various systems. Furthermore, tools like as compilers and cross-platform development environments are available to assist developers in creating platform-independent C++ code. In machine learning, derivatives are used to optimize models and improve performance. Gradient descent is a popular optimization algorithm that uses derivatives to find the minimum of a function.

This Grade work will look at one of the most significant aspects of the C++ programming language, because it allows researchers to forecast how a system will act in the future based on its current state, the derivative statement is especially valuable in simulators. This is critical for many applications, such as forecasting weather patterns or simulating disease propagation. Researchers can develop more accurate predictions and gain a better understanding of the underlying mechanisms driving the system's behavior by using the derivative statement.

Online GDB is a web-based compiler and debugger for C, C++, and other programming languages. It allows users to write, compile, and run code directly from their web browser, without the need for any additional software or tools. Thus, the grade work focuses.

CHAPTER 1. PROBLEM STATEMENT

1.1. Problem Statement

The problem statement for derivatives in C++ involves calculating the rate of change of a function at a specific point or over a range of values. This can be done using various methods such as forward, backward, and central difference approximations. The goal is to create an algorithm that accurately approximates the derivative of the function, while also being efficient and easy to use. The algorithm should be able to handle a wide range of input values and provide accurate results, even in cases where the function is complex or difficult to evaluate. This problem statement is relevant in many fields, including physics, engineering, finance, and computer science, and can be used to solve a wide range of real-world problems.

Derivatives in C++ can be difficult to implement for a variety of reasons. One of the most difficult issues is determining the best approach for approximating the derivative, as different methods may be more or less accurate depending on the function under consideration. Another problem is ensuring that the algorithm is efficient and capable of dealing with a wide range of input values while still producing accurate outputs. When working with complex or highly nonlinear functions, this can be very problematic. Furthermore, it is possible to make mistakes when implementing the method, which can result in inaccurate results or even crashes. Finally, grasping the mathematical concepts underlying derivatives and how they might be implemented in C++ requires a learning curve. Overall, while C++ variants can be powerful tool for solving complex problems.

CHAPTER 2. THEORETICAL PART

2.1. Derivative statement

For the distance course "Derivatives" you must build a simulator. The main point of the work includes:

- highlight the C++ language.
- explain the fundamental concepts of the topic Derivatives.
- create a derivative for the simulator and build a block diagram.
- explain the programming language and technologies used in building the program.

2.2. The purposes of derivative

What are derivatives?

Derivatives are used to find the rate of changes of a quantity with respect to other quantity.

How derivative function works in C++?

The **derivative** of a function is the rate at which the function value is changing, with respect to *x*, at a given value of x. Graphically we can say that derivative is nothing but a slope at a particular point of a function. The slope of the tangent line differs from one point to the next. The value of the derivative of a function depends on the point in which we decide to evaluate it. Basically, we often mention the slope of a function instead of the slope of its tangent lines.

Notation

- The representation of a derivative is by a prime symbol. Example $f'(x)$ represents the derivative of a function f evaluated at point x. Uniformly, writing $(3x + 2)'$ indicates we are carrying out the derivative of the function $3x + 2$. The prime symbol disappears as soon as the derivative has been calculated.

A derivative is a measure of how much a function changes with respect to its input. It is defined as the limit of the ratio of the change in the function to the change in the input, as the change in the input approaches zero. In other words, the derivative

of a function $f(x)$ at a point x is defined as: $f'(x) = lim(h -> 0) [f(x + h) - f(x)] / h$ where h is a small positive number that represents the change in x.

In C++, we can use numerical methods to approximate the derivative of a function at a point, since the limit definition of the derivative is often difficult to compute exactly. One common numerical method for approximating derivatives is the finite difference method, which uses the slope of a secant line between two points on the function to approximate the derivative.

To use the finite difference method to approximate the derivative of a function f(x) at a point x, we choose a small value of h and compute the slope of the secant line between the points $(x, f(x)) and (x + h, f(x + h))$. This slope is given by: $[f(x + h) - f(x)]$ / h which is an approximation of the derivative of $f(x)$ at x. In C++, we can define a function that represents the function we want to differentiate, and then use the finite difference method to approximate the derivative of that function at a given point. We can also use other numerical methods to approximate derivatives, such as Taylor series expansions, Runge-Kutta methods, and more advanced techniques. Overall, numerical methods are a powerful tool for approximating derivatives in C++, and they are widely used in scientific and engineering applications.

Steps for finding a derivative function in C++includes.

1. Identify the variable terms and constant terms in the equation.
2. Multiply the coefficients of each variable term by their exponents.
3. Decrement of each exponent by one.
4. Use new coefficient in place of the old ones.
5. Find the value of the function from the value of the value.

Using the procedures, we can find the derivatives of a function but using code, but using code to do so can be a bit more challenging.

Derivatives of usual function

We are going to have a look at the list of the most important derivatives.

Although these formulas can be formally proven, it is a great idea to fully understand how derivatives work.

a. The Constant function

The derivative of constant function $f(x) = c$ is always zero since the function does not change with respect to its input.

The derivative of a function is a measure of how much the function changes with respect to its input. In general, the derivative of a function f(x) at a point x is defined as.

$'(x) = lim(h -> 0) [f(x + h) - f(x)] / h$

where h is a small positive number that represents the change in x.

When we apply this definition to a constant function, we get:

$f'(x) = lim(h -> 0) [f(x + h) - f(x)] / h$

$= lim(h -> 0) [3 - 3] / h$

$= lim(h -> 0) 0 / h$

$= 0$

Also,$(8)' = 0$

So, the derivative of a constant function is always zero. This makes sense, since a constant function does not change with respect to its input, so the rate of change (i.e. the derivative) is always zero.

b. the identity function

The identity function is a function that returns its input as its output. In other words, the output of the function is equal to the input of the function. The identity function is denoted by the symbol "id" or "I", and is defined as:

$id(x) = x$

For any value of x, the output of the function is just x itself. For example, if we plug in $x = 8$, we get:

$id(8) = 8$

Similarly, if we plug in $x = 5$, we get:

$id(5) = 5$

The identity function is important because it is the simplest example of a function. It is also an example of a function that is both injective and surjective. An injective function is a function where every input has a unique output, while a surjective function is a function where every output has at least one input that maps to it. Since the identity function maps every input to a unique output and every output has at least one input that maps to it, it is both injective and surjective.

A function of form x^n

The derivatives of function $f(x)$ is a measure how much the function changes with respect to its input x. In general, the derivative of a function $f(x)$, and is also defined as;

$'(x) = lim(h -> 0) [f(x + h) - f(x)] / h$

Where h is a small positive number which represents the change in x.

When we apply this definition to a function of the form x^n, we get;

$(x) = x^n f'(x) = lim(h -> 0) [(x + h)^n - x^n] / h$

Notation

- The rule stated above applies to all types of exponents. it can natural, fraction and whole. It is always important that this exponent is constant. There is another rule which is exponential function.
- It is often the case that a function satisfies this form requires a bit of reformulation before proceeding to the derivative. It is the case of roots (square, cubic) representing fractional exponents.

An exponential function (form a^x with a>0)

To implement an exponential function of the form a^x with a > 0 in C++. You can use the built in library function pow (). The pow function takes two arguments: the base and exponent x and the return the value a^x.

Here's an example of how you can use the pow() function to implement an exponential function:

```c++
#include <iostream>
#include <cmath>

double exponential(double a, double x) {
    return pow(a, x);
}

int main() {
    double a = 2.0;
    double x = 3.0;
    double result = exponential(a, x);
    std::cout << "The exponential of " << a << " to the power of " << x << " is " << result << std::endl;
    return 0;
}
```

In this example, we define a function called exponential() that takes two arguments: the base a and the exponent x. The function uses the pow() function to compute the value of a^x, and returns the result.

In the main() function, we call the exponential() function with a = 2.0 and x = 3.0, and store the result in a variable called result. We then print out the result using std::cout.

This program will output:

The exponential of 2 to the power of 3 is 8

You can change the values of a and x to compute different exponential functions just to get the understanding of this.

The function e^x in C++

To compute the derivative of the function f(x) = e^x in C++, you can use the built-in math library function exp(). The exp() function takes one argument, the exponent x, and returns the value e^x.

Here's an example of how you can use the exp() function to compute the derivative of the function f(x) = e^x:

```c++
#include <iostream>
#include <cmath>

double derivative(double x) {
    return exp(x);
}

int main() {
    double x = 2.0;
    double h = 0.0001;
    double result = (derivative(x + h) - derivative(x)) / h;
    std::cout << "The derivative of e^x at x = " << x << " is " << result << std::endl;
    return 0;
}
```

```

In this example, we define a function called derivative() that takes one argument, the exponent x. The function uses the exp() function to compute the value of e^x, and returns the result.

In the main() function, we call the derivative() function with x = 2.0, and store the result in a variable called result. We then compute the derivative of the function using the formula (f(x + h) - f(x)) / h, where h is a small number (0.0001 in this case) that represents the step size. We print out the result using std::cout.

This program will output:

\

he derivative of e^x at x = 2 is 7.38906

Basic Rules of Derivatives

Power rule: This rule is used when we need to find the derivative of a function that has a variable raised to a power. The power rule states that if f(x) = x^n, then f'(x) = nx^(n-1). In C++, we can implement this rule using the pow() function from the <cmath> library. The pow() function takes two arguments: the base and the exponent, and returns the base raised to the exponent. Here's an example implementation.

```c++
double power_rule(double x, double n) {
   return n * pow(x, n-1);
}
```

2. Product rule: This rule is used when we need to find the derivative of a function that is the product of two other functions. The product rule states that if f(x) = u(x) * v(x), then f'(x) = u'(x) * v(x) + v'(x) * u(x). In C++, we can implement this
```

rule using the product rule formula. Here's an example implementation:

```c++
double product rule(double u, double v, double u_prime, double v_prime) {
   return u_prime * v + v_prime * u;
}
```

3. Quotient rule: This rule is used when we need to find the derivative of a function that is the quotient of two other functions. The quotient rule states that if f(x) = u(x) / v(x), then f'(x) = (u'(x) * v(x) - v'(x) * u(x)) / v(x)^2. In C++, we can implement this rule using the quotient rule formula. Here's an example implementation:

```c++
double quotient_rule(double u, double v, double u_prime, double v_prime) {
   return (u_prime * v - v_prime * u) / pow(v, 2);
}
```

4. Chain rule: This rule is used when we need to find the derivative of a function that is composed of two or more other functions. The chain rule states that if f(x) = g(h(x)), then f'(x) = g'(h(x)) * h'(x). In C++, we can implement this rule using the chain rule formula. Here's an example implementation:

-
  - ``c++ double chain_rule(double g, double h, double g_prime, double h_prime)
-

How does composite function works??

A composite function is a function that includes another function. A composite function is one that can be broken down into many components, each of which is a function in and of itself, and these parts are not linked by addition, subtraction,

product, or division.

To understand how composite functions work, let's consider two functions f(x) and g(x). The function g(x) takes an input x and produces an output g(x). The function f(x) takes an input f(x) and produces an output f(g(x)). So, when we evaluate the composite function (f o g)(x), we first evaluate g(x) to get a value, say a, and then evaluate f(a) to get the final output.

Here's an example to illustrate how composite functions work. Let's consider the functions f(x) = x^2 and g(x) = 2x + 1. To evaluate the composite function (f o g)(x), we first need to evaluate g(x) to get a value, say a. So, if we plug in x = 2 into g(x), we get:

g(2) = 2(2) + 1 = 5

Now, we evaluate f(a) to get the final output:

f(5) = 5^2 = 25

So, (f o g)(2) = f(g(2)) = f(5) = 25.

In summary, composite functions allow us to combine two functions to create a new function, where the output of one function becomes the input of the other function. To evaluate a composite function, we first evaluate the inner function, and then use the output as the input to the outer function.

The chain rule

The chain rule of derivatives is a rule used to calculate the derivative of function composed of two or more functions.

In C++, the chain rule can be implemented using the concept of function composition.

Let's consider a function f(x) = g(h(x)), where g and h are functions of x. To find the derivative of f(x), we need to use the chain rule, which states that: f'(x) = g'(h(x)) * h'(x) This means that the derivative of f(x) is equal to the derivative of g evaluated at h(x), times the derivative of h(x). In C++, we can implement this rule using the following steps:

1. Define the functions g and h as separate functions, each taking an input of type double and returning a value of type double.

2. Define a new function f that takes an input of type double and returns a value of type double. This function should call the functions g and h using the input x, and return the result of g(h(x)).

2. Define a new function f_prime that takes an input of type double and returns a value of type double. This function should use the chain rule to calculate the derivative of f at x. Specifically,

it should calculate g'(h(x)) and h'(x), and then multiply them together to get f'(x). Here's an example implementation of the chain rule of derivatives in

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

double g(double x) {
    return std::sin(x);
}

double h(double x) {
    return std::cos(x);
}
```

```
double f(double x) {
    return g(h(x));
}

double f_prime(double x) {
    double h_prime = -std::sin(x);
    double g_prime = std::cos(h(x));
    return g_prime * h_prime;
}

int main() {
    double x = 1.0;
    std::cout << "f(" << x << ") = " << f(x) << std::endl;
    std::cout << "f'(" << x << ") = " << f_prime(x) << std::endl;
    return 0;
}
```

In this example, the functions g and h are defined as the sine and cosine functions, respectively. The function f is defined as the composition of g and h, and the function f_prime uses the chain rule to calculate the derivative of f at a given point x. The main function simply calls f and f_prime fo

CHAPTER 3. PRACTICAL PART

3.1. Online Gdb for C++

Online GDB is a web-based compiler and debugger for C++ that allows you to write, compile, and debug your code in one place. It is a convenient tool for developers who do not want to install a compiler on their local machine or who want

to share their code with others.

When you first navigate to the Online GDB website, you will see a simple interface with a text editor, a console, and several buttons. To use Online GDB with C++, you need to select the "C++" language option from the drop-down menu in the top left corner.

Once you have selected the C++ language, you can begin writing your code in the editor window. You can create new files or use existing ones by clicking the "New File" or "Open File" buttons. As you write your code, you can use the console to test your code and view the output.

To compile your code, click the "Compile" button in the top menu. If there are any errors, they will be displayed in the "Error" tab. You can click on the error message to jump to the relevant line in your code.

To debug your code, click the "Debug" button in the top menu. This will open a new window with the debugger interface. Here, you can set breakpoints, step through your code, and view the values of variables. The debugger interface is like what you would see in a desktop IDE, with buttons for stepping over, stepping into, and stepping out of functions.

Once you have finished debugging your code, you can exit the debugger and return to the editor window by clicking the "X" button in the top right corner.

To run your code, click the "Run" button in the top menu. This will execute your code and display the output in the "Output" tab. You can also input values into the console to test your code with different inputs.

Online GDB also provides other useful features, such as the ability to save and share your code with others, and the ability to download your code as a file.

Overall, Online GDB is a powerful and convenient tool for C++ developers who want to write, compile, and debug their code in a web-based environment. Its simple interface and powerful features make it a great choice for developers of all skill levels.

GNU Debugger

The GNU Debugger (GDB) is a powerful command-line tool that allows developers to debug programs written in various programming languages, including C, C++, Ada, and others. GDB is a free and open-source tool that is available on most Unix-based systems, including Linux, macOS, and FreeBSD.

GDB enables developers to monitor and control program execution, set breakpoints, explore memory and variables, and much more. GDB allows developers to step through their code line by line, analyze variable and expression values, and view the call stack.

GDB can be utilized in a variety of ways, depending on the developer's requirements. GDB, for example, can be launched in command-line mode, which allows the developer to communicate with GDB via a terminal window. GDB can also be linked into a linked Development Environment (IDE), such as Eclipse or Code::Blocks, where the developer can interact with GDB via a graphical interface

One of GDB's most useful features is the ability to establish breakpoints in code. A breakpoint is a point in the code when GDB will interrupt program execution and allow the developer to inspect the program's state. Breakpoints can be placed on a single line of code, a function call, or a condition.

GDB also includes several instructions for inspecting the program's state. The "print" command, for example, can be used to print the value of a variable or expression. The "info" command displays program information such as the call stack and the current line of code.

Overall, GDB is a robust and adaptable debugging tool. While it can be challenging to use at first, GDB is an essential tool for any developer who needs to debug their code.

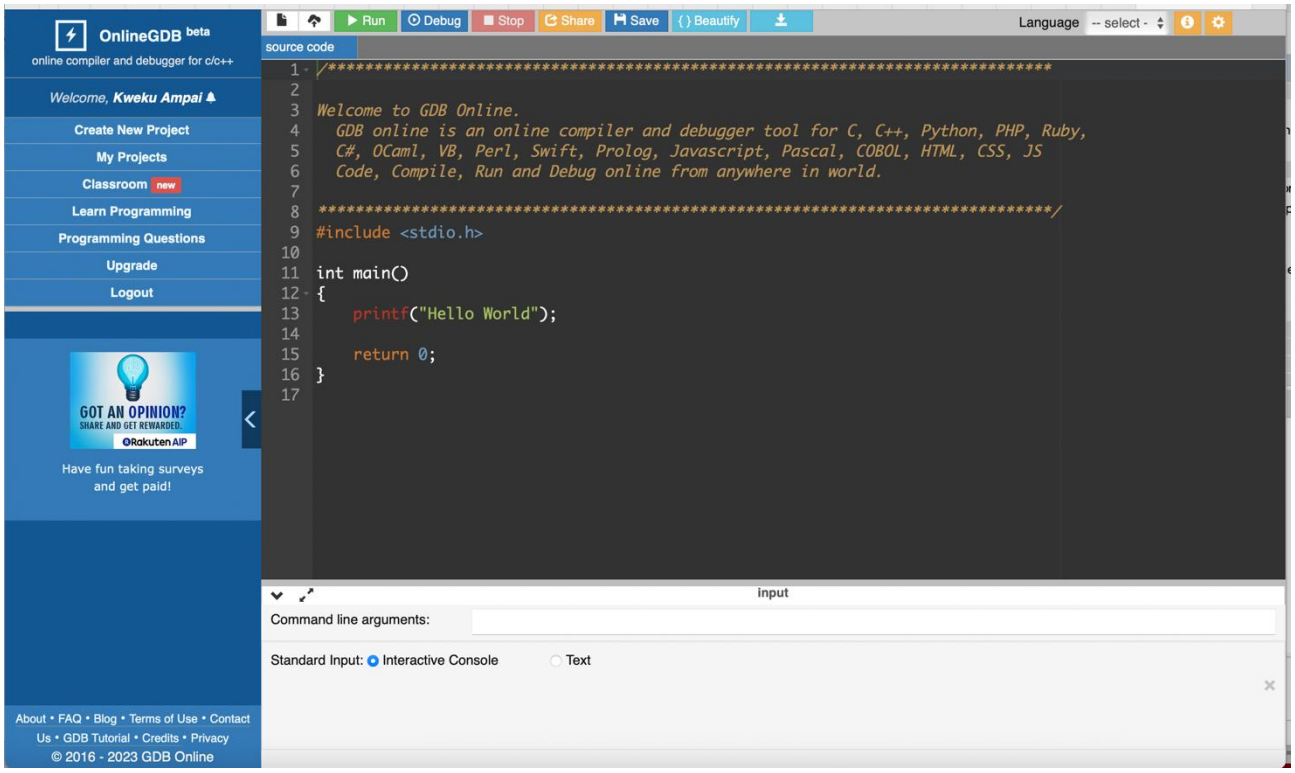

Figure 1. GDB GNU Builder

3.2. Components of OnlineGDB

OnlineGDB is a web-based compiler and debugger with a variety of developer-friendly features. OnlineGDB's primary components include the following:

1. Web-based code editor: OnlineGDB includes a web-based code editor that allows developers to write code in a variety of programming languages such as C, C++, Java, Python, and others.

2. Compiler: OnlineGDB comes with a compiler that can compile code written in a variety of computer languages. The compiler is built within the web-based interface, allowing developers to build code with a single click.

3. Debugger: OnlineGDB includes a debugger that allows developers to step through their code, set breakpoints, and inspect the program's state. The debugger is built within the web-based interface, allowing developers to debug their code in the same environment.

4. Console: OnlineGDB has a console that allows developers to interact with

the input and output of their program. The console is incorporated within the web-based interface, allowing developers to examine the output of their application while still working in the same environment.

5. Sharing: By providing a URL to their code, developers can share their code with others. This can be beneficial for interacting with others or obtaining debugging assistance.

Overall, OnlineGDB is a robust and adaptable solution for web-based developers who need to write, compile, and debug code.

3.3. The Application of Derivative in C++

Derivatives in C++ are used in a variety of domains, including computer graphics, optimization, physics, and engineering. The following are some examples of how derivatives are utilized in C++:

1. Computer graphics: Derivatives can be used to calculate the slope of a curve at a specific location, which is important in designing smooth curves and surfaces in computer graphics.

2. Optimization: In optimization issues, derivatives can be employed to identify the least or maximum value of a function.

3. Physics: Derivatives can be used to compute an object's velocity and acceleration, which is useful in simulating the motion of objects in physics.

4. Engineering: Derivatives can be used to calculate a system's rate of change, which is useful in engineering for modelling and analyzing complex systems

Derivatives in C++ can be calculated using a variety of numerical approaches, including the finite difference method and automatic differentiation. Libraries like Boost and GSL include functions for calculating derivatives, while C++ frameworks like ROOT and OpenCV make considerable use of derivatives in their algorithms.

Example 1.

```cpp
```

```
#include <boost/math/differentiation/autodiff.hpp>
#include <iostream>

using namespace boost::math::differentiation;

// Define the function to differentiate
auto f(auto x) {
    return sin(x) + 2 * x;
}

int main() {
    // Calculate the first derivative of f at x = 1
    auto x = make_fvar<double, 1>(1);
    auto result1 = derivative(f, wrt(x));

    // Calculate the second derivative of f at x = 1
    auto result2 = derivative(result1, wrt(x));

    // Print the results
    std::cout << "f'(1) = " << result1 << std::endl;
    std::cout << "f''(1) = " << result2 << std::endl;

    return 0;
}
```

``` In this example, we first calculate the first derivative of `f` at `x = 1`, and then we calculate the second derivative of `f` at `x = 1` by calling the `derivative` function again, passing in the result of the first derivative as the function to differentiate. Finally, we print the results to the console. In this case, the output would be:
```

```
f'(1) = 2.5403
f''(1) = -0.416147
```

which are the approximate values of the first and second derivatives of `f` at `x = 1`.

Example 2.

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

using namespace std;

// Define the function to differentiate
double f(double x) {
    return sin(x) + 2 * x;
}

// Calculate the derivative of f at x using the finite difference method
double derivative(double x, double h) {
    double fx = f(x);
    double fxh = f(x + h);
    return (fxh - fx) / h;
}

int main() {
    // Calculate the derivative of f at x = 1 using h = 0.001
```

```
    double x = 1;
    double h = 0.001;
    double result = derivative(x, h);

    // Print the result
    cout << "f'(1) = " << result << endl;

    return 0;
}
```

n this example, the `f` function represents the function whose derivative we want to calculate, and the `df` function represents the derivative of `f` using the finite difference method. The `main` function calls the `df` function with a value of `x` and a small value of `h` to calculate the derivative of `f` at `x`. The result is then printed to the console.

Note that this is a simple example and that there are many other methods for calculating derivatives that are more accurate and efficient.

Example 3.

```cpp
#include <iostream>
#include <complex>

using namespace std;

complex<double> f(complex<double> z)
{
    return sin(z);
}
```

```
complex<double> df(complex<double> z, double h)
{
   complex<double> hI(0.0, h);
   return imag(f(z + hI)) / h;
}

int main()
{
   complex<double> z(1.0, 1.0);
   double h = 0.0001;

   complex<double> result = df(z, h);

   cout << "The derivative of sin(z) at z = " << z << " is " << result << endl;

   return 0;
}
```

In this example, the `f` function represents the complex function whose derivative we want to calculate, and the `df` function represents the derivative of `f` using the complex-step differentiation method. The `main` function calls the `df` function with a value of `z` and a small value of `h` to calculate the derivative of `f` at `z`. The result is then printed to the console.

Notation

the complex-step differentiation method is a technique for calculating the derivative of complex functions that is more accurate than the finite difference method, especially for functions that have a complex output.

Example 4.

```cpp
#include <iostream>
#include <cmath>

using namespace std;

double f(double x)
{
    return sin(x);
}

double df(double x, double h)
{
    return (f(x + h) - f(x)) / h;
}

int main()
{
    double x = 0.5;
    double h = 0.0001;

    double result = df(x, h);

    cout << "The derivative of sin(x) at x = " << x << " is " << result << endl;

    return 0;
```

Example 5.

```
#include <bits/stdc++.h>
```

```
using namespace std;
long long dTerm(string t1, long long v)
{

  string coeffStr = "";
  int i;
  for (i = 0; t1[i] != 'x'; i++)
    coeffStr.push_back(t1[i]);
  long long coeff = atol(coeffStr.c_str());
  string powStr = "";
  for (i = i + 2; i != t1.size(); i++)
    powStr.push_back(t1[i]);
  long long expo = atol(powStr.c_str());
  return coeff * expo * pow(v, expo - 1);
}
long long dVal(string& poly, int v)
{
  long long ans = 0;
  istringstream is(poly);
  string t1;
  while (is >> t1) {
    if (t1 == "+")
      continue;
      else
      ans = (ans + dTerm(t1, v));
  }
cout<<"The derivative of a function is "<< ans;
}
int main()
```

```
{
  string str = "2x^2 + 4x^1";
  int v = 2;
  cout << dVal(str, v);
  return 0;
}
```

OUTPUT

The derivative of a function is 12.

Example 6.

```
#include <iostream>
#include <iomanip>
#include <boost/multiprecision/cpp_dec_float.hpp>
#include <boost/math/constants/constants.hpp>


int main(int, char**)
{
   using boost::math::constants::pi;
   using boost::multiprecision::cpp_dec_float_50;
   //
   // We'll pass a function pointer for the function object passed to derivative,
   // the typecast is needed to select the correct overload of std::sin:
   //
   const float d_f = derivative(
      pi<float>() / 3,
      0.01F,
```

```
  static_cast<float(*)(float)>(std::sin)
);

const double d_d = derivative(
  pi<double>() / 3,
  0.001,
  static_cast<double(*)(double)>(std::sin)
  );
//
// In the cpp_dec_float_50 case, the sin function is multiply overloaded
// to handle expression templates etc.  As a result it's hard to take its
// address without knowing about its implementation details.  We'll use a
// C++11 lambda expression to capture the call.
// We also need a typecast on the first argument so we don't accidentally pass
// an expression template to a template function:
//
const cpp_dec_float_50 d_mp = derivative(
  cpp_dec_float_50(pi<cpp_dec_float_50>() / 3),
  cpp_dec_float_50(1.0E-9),
  [](const cpp_dec_float_50& x) -> cpp_dec_float_50
  {
    return sin(x);
  }
  );

// 5.000029e-001
std::cout
  << std::setprecision(std::numeric_limits<float>::digits10)
  << d_f
```

```
    << std::endl;

  // 4.999999999998876e-001
  std::cout
    << std::setprecision(std::numeric_limits<double>::digits10)
    << d_d
    << std::endl;
  // 4.99999999999999999999999999999999999999999999999e-01
  std::cout
    << std::setprecision(std::numeric_limits<cpp_dec_float_50>::digits10)
    << d_mp
    << std::endl;
}
```

The derivative anticipated value is 0.5. In this case, the central difference rule is ill-conditioned, which means it has a minor loss of precision. Keeping this in mind, the results are consistent with the expected value of 0.5.

Example 7.

Input: str = "2x^3 +1x^1 + 3x^2"

  val = 2

Output: 37

Explanation: 6x^2 + 1x^0 + 6x^1

  Putting x = 2

  6*4 + 1 + 6*2 = 24 + 1 + 12 = 37

Input: str = “1x^3”

  val = 2

Output: 12

Explanation: 1 * 3 *x^2

  Putting x = 2

3 * 4 = 12

3.4. Design And Programming of a Simple Guessing Game

Information regarding my grade work, which includes screenshots.

Finally, I have designed a simple guessing game in C++ with the help of onlinegdb. A guessing game in C++ involves the user guessing a randomly generated number within a certain range. The program will prompt the user to enter a number, and then compare it to the generated number. If the guess is too high or too low, the program will give the user a hint and prompt them to guess again. The game continues until the user has correctly guessed the number, at which point the program will congratulate them and ask if they want to play again.

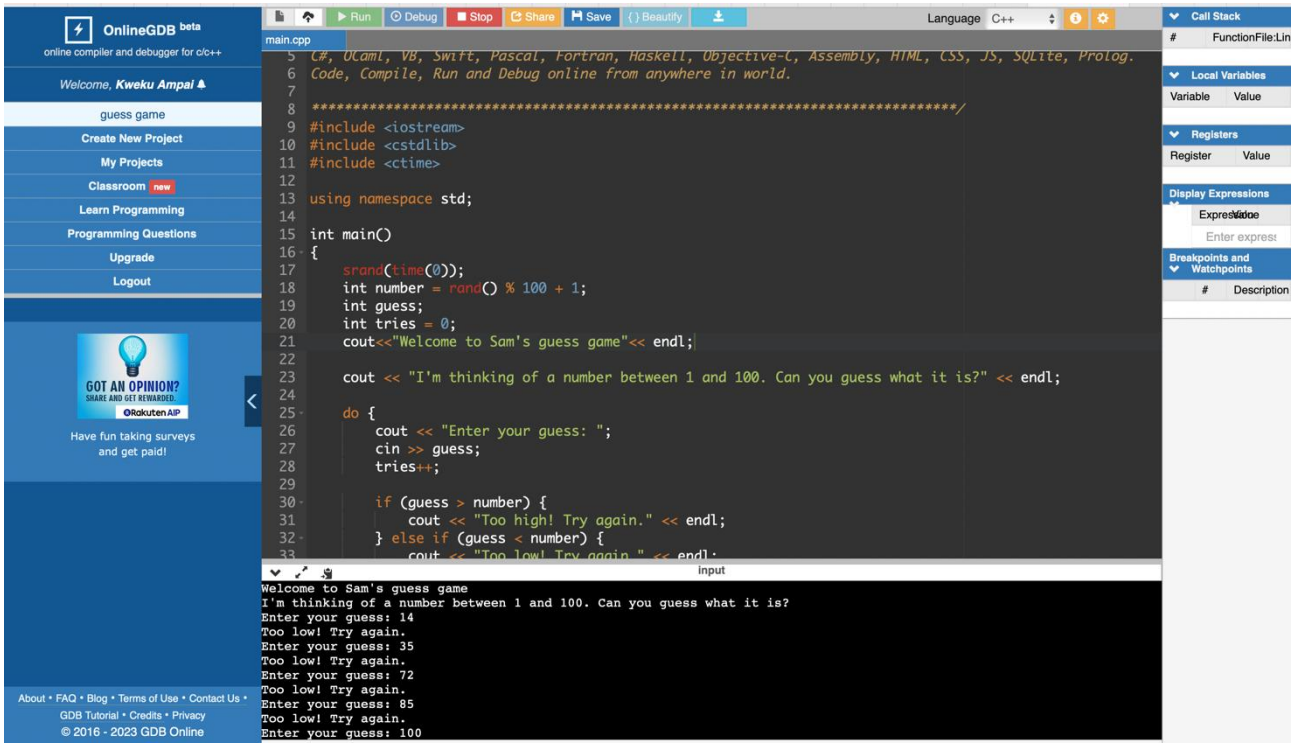

Figure 1.1

The guess game.

CONCLUSIONS

There was independent consideration of several sources of information and mastery of the project in this Grade work assignment. Consolidation of the theoretical expertise offered by lecture content I have studied at the prestigious university.

Derivatives in C++ can be challenging to implement, but if mastered, they can be a useful tool for tackling complex mathematical problems. While there are numerous approaches for approximating derivatives, the optimum strategy will depend on the specific problem at hand. C++ gives you a lot of flexibility and control over how these methods are implemented, but it's also quite easy to make mistakes if you're not careful. Overall, studying derivatives in C++ can be a difficult but rewarding subject for anyone interested in mathematics, science, or engineering.

REFERENCES

1. COMPUTER SOFTWARE DERIVATIVE WORKS**:** THE CALM BEFORE THE STORM

2. MODELLING DERIVATIVES APPLICATION: JUSTIN LONDON

3. OPTIONS AND DERIVATIVES PROGRAMMING IN C++: CARLOS OLIVEIRA MATH FOR PROGRAMMERS

APPENDIX A. Algorithm translation

Descriptive Algorithm.

Step 1. Open your web browser for example Safari web browser.

Step 2. Type in your search engine OnlineGDB compiler.

Step 3. Change the language to C++.

Step 4. Type in the codes for the guess game.

Step 5. Run the code.

Step 6. Play the guess game.

APPENDIX B. Program code

```
/*******************************************************************
***************

Welcome to GDB Online.
GDB online is an online compiler and debugger tool for C, C++, Python, Java,
PHP, Ruby, Perl,
C#, OCaml, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly,
HTML, CSS, JS, SQLite, Prolog.
Code, Compile, Run and Debug online from anywhere in world.

*******************************************************************
*****************/
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
   srand(time(0));
   int number = rand() % 100 + 1;
   int guess;
   int tries = 0;
   cout<<"Welcome to Sam's guess game"<< endl;

   cout << "I'm thinking of a number between 1 and 100. Can you guess what it
is?" << endl;
```

```
        do {
            cout << "Enter your guess: ";
            cin >> guess;
            tries++;

            if (guess > number) {
                cout << "Too high! Try again." << endl;
            } else if (guess < number) {
                cout << "Too low! Try again." << endl;
            } else {
                cout << "Congratulations! You guessed the number in " << tries << " tries." << endl;
            }
        } while (guess != number);

        return 0;
    }
```