

Grade 10 Computer Science (20S)

A Course for
Independent Study

GRADE 10
COMPUTER SCIENCE (20S)

*A Course for
Independent Study*

2010

Manitoba Education

Manitoba Education Cataloguing in Publication Data

Grade 10 computer science (20S) : a course for independent study

ISBN-13: 978-0-7711-4593-3

1. Computer science—Study and teaching (Secondary).
 2. Computer science—Programmed instruction.
 3. Computer science—Study and teaching (Secondary)—Manitoba.
 4. Electronic data processing—Study and teaching (Secondary).
 5. Electronic data processing—Programmed instruction.
 6. Electronic data processing—Study and teaching (Secondary)—Manitoba.
- I. Manitoba. Manitoba Education.
004

Copyright © 2010, the Government of Manitoba, represented by the Minister of Education.

Manitoba Education
School Programs Division
Winnipeg, Manitoba, Canada

Every effort has been made to acknowledge original sources and to comply with copyright law. If cases are identified where this has not been done, please notify Manitoba Education. Errors or omissions will be corrected in a future edition. Sincere thanks to the authors and publishers who allowed their original material to be used.

All images found in this document are copyright protected and should not be extracted, accessed, or reproduced for any purpose other than for their intended educational use in this document.

Any websites referenced in this document are subject to change. Educators are advised to preview and evaluate websites and online resources before recommending them for student use.

Acknowledgements

Manitoba Education and Youth gratefully acknowledges the contributions of the following individuals in the development *Grade 10 Computer Science (20S): A Course for Independent Study*.

Course Writer:

Brent Giesbrecht	Writer	Green Valley School Hanover School Division
------------------	--------	--

Manitoba Education and Youth Staff

School Programs Division

Greg Backhouse	Independent Study Option Supervisor	Distance Learning and Information Technologies Unit Instruction, Curriculum and Assessment Branch
Louise Boissonneault	Publications Editor	Document Production Services Instruction, Curriculum and Assessment Branch
Lee-Ila Bothe	Coordinator	Document Production Services Instruction, Curriculum and Assessment Branch
Darryl Gervais	Project Leader	Distance Learning and Information Technologies Unit Instruction, Curriculum and Assessment Branch
Gilles Landry	Project Manager	Distance Learning and Information Technologies Unit Instruction, Curriculum and Assessment Branch
Lindsay Walker	Desktop Publisher	Document Production Services Instruction, Curriculum and Assessment Branch

Notes

Contents

Acknowledgements *iii*

Introduction *1*

Module 1: Introduction to Computer Science

Introduction	3
Lesson 1: Language and Machines	5
Lesson 2: Circuits, Switches, and Tubes	11
Lesson 3: Windows, Objects, and the Web	15
Assignment 1.1: History Research Project	21
Lesson 4: Structured Programming	23
Lesson 5: Ethics in Computer Technology	37
Assignment 1.2: Programming and Ethics	38
Lesson 6: Career Opportunities in Programming	43
Assignment 1.3: Careers	49
Answer Key	

Module 2: Introduction to Programming

Introduction	3
Lesson 1: Components of Computers	5
Lesson 2: Building a Program	9
Lesson 3: Introduction to Debugging	17
Lesson 4: Parts of a Program	21
Lesson 5: Variables, Basic Data Types, and Data Storage	31
Lesson 6: C++ Statements	41
Assignment 2.1: Metric Conversion: Temperature	55
Assignment 2.2: The Jogger	56
Assignment 2.3	57
Answer Key	

Module 3: Control Structures

Introduction	3
Lesson 1: Flow Control and Booleans	5
Lesson 2: <code>if</code> Statements	13
Assignment 3.1: Net Income	22

Assignment 3.2: Making Change 23

Lesson 3: `switch` Statement 25

Assignment 3.3 30

Answer Key

Module 4: Loops: Going in Circles!

Introduction 3

Lesson 1: The `while` Loop 5

Lesson 2: Event-Controlled `while` Loop 9

Assignment 4.1: Falling 11

Lesson 3: The `for` Loop 13

Lesson 4: Nested `for` Loops 17

Assignment 4.2: Multiplication 20

Lesson 5: The `do-while` Loop 21

Assignment 4.3 24

Answer Key

Module 5: Functions: Piecework Programming

Introduction 3

Lesson 1: What Is a Function? 5

Lesson 2: Execution Flow of Functions 13

Lesson 3: Function Parameters 23

Lesson 4: Value and Reference Parameters 25

Lesson 5: Scope of Variables 31

Lesson 6: Passing Data to Functions 35

Lesson 7: Returning Single Data Values 37

Lesson 8: Returning Multiple Data Values 41

Lesson 9: Function Documentation 47

Lesson 10: A Complete Example! 49

Lesson 11: Functions: A Summary 53

Assignment 5.1: Calculating Box Dimensions 58

Assignment 5.2: Employee Pay Calculator 59

Assignment 5.3 61

Answer Key

Glossary

Grade 10 Computer Science

Introduction

Welcome to *Grade 10 Computer Science: A Course for Independent Study*. The purpose of this Introduction is to help you become familiar with the course. It's like the first few days of school, when the teacher tells you about the course, and what you are going to learn.

What is Computer Science?

Computers are everywhere. They are in homes, schools, and businesses. Microprocessors are at the core of every computer. Microprocessors are also part of things like video games, cellphones, televisions, and cars. Computer science is about designing and controlling microprocessors. A major part of computer science (and this course) is computer programming.

Computer programming is telling a computer what to do in a language that it can understand. One of those languages is Visual C++, the one that you will be using in this course. However, computer scientists do more than programming. Some conduct research in areas like artificial intelligence (trying to understand whether machines can think like people). Some work on computer networks or try to make computers run better and faster. Computer scientists work for many different kinds of organizations, large or small.

What This Course is About

This is an optional, full credit entry-level programming course. It will give you the chance to solve problems, accomplish tasks, and express creativity by using programming techniques in Visual C++. The techniques that you will learn are applicable to other programming languages. This course will enable you to explore and further develop skills in solving problems and prepare you for further study at college or university.

What You Will Need

- . A computer with the following **minimum** requirements:
 - a 1.6 GHz processor
 - 1024 MB of memory
 - 3 GB of hard disk space
 - **Windows XP** operating system (SP3)
 - a **word processor**
 - an **email** program and **email address** so that you can email your assignments to your tutor/marker
- . Access to the Internet and the ability to download and to activate registration keys so you can download the following:
 - Microsoft Visual C++ 2010 Express, available at <www.microsoft.com/express/Downloads/#2010-Visual-CPP>.
(Note: Microsoft Visual Studio 2010 Express requires that you register your evaluation copy **within 30 days of installation**. Once you have entered your Registration Key, the product is free.)
 - Course data files available on the Manitoba Education website at <www.edu.gov.mb.ca/dl/downloads>.

Note: ISO does not provide technical support for hardware-related issues; consult your tutor/marker for software support. If further trouble shooting is required, you may have to consult the software manufacturer or a professional computer technician.

What if You Need Help?

Taking an independent study course is different from taking a course in a classroom. Instead of the teacher telling you to complete an exercise, you need to tell yourself. However, there are two people who can help you be successful in your course. They are:

Your Tutor/Marker

The first person who can help you is your tutor/marker. Tutor/markers are experienced teachers who tutor independent study students and mark assignments and exams. When you are having difficulty, be sure to contact your tutor/marker. They are there to help you. You should feel free to contact your tutor/marker at any time during this course. Your tutor/marker's name, email address, and telephone number would have been provided to you on the form called a "Record of Progress" that was mailed with your course. It is a white paper that was not enclosed inside the wrapped course. If you need assistance to locate the name and contact information for your tutor/marker, you can contact:

Distance Learning Unit
555 Main Street
Winkler, MB R6W 1C4
1-800-465-9915

Your Study Partner

The next person who can help you with your course is your study partner. A study partner is **someone you choose** who will help you learn. It may be someone who knows something about computer science, but it doesn't have to be. A study partner could be someone else who is taking this course, a teacher, parent, sibling, friend, or anybody else who can help you. Most importantly, a study partner should be someone you feel comfortable with and will help you with the course. Your study partner can help you keep on schedule, check your work, help you make sense of assignments, and give you advice.

How is the Course Organized?

The course is broken down into five modules.

- Module 1 — Introduction to Computer Science
- Module 2 — Introduction to Programming
- Module 3 — Control Structures
- Module 4 — Loops: Going in Circles!
- Module 5 — Functions: Piecework Programming
- Glossary

Each module contains several lessons. A typical lesson includes content that you need to read. Each module also includes learning activities to help you try out what you have read. Since you do not hand in your learning activities, you can check your own answers with the answer keys at the end of each module.

Each module also includes three assignments that you send to your tutor/marker. At the end of Module 3, you will write your midterm exam. At the end of Module 5, you will write your final exam.

The glossary at the end of the course contains definitions of important terms.

How Do You Know How Well You Are Learning?

You will know how well you are learning by how well you complete the following:

- 15 Assignments: 60% (each assignment is worth 4% of the final mark of the course). You will be mailing or emailing each assignment to your tutor/marker. For assignments that include computer code, you must send the code to your tutor/marker, either by emailing it, or by mailing it on a CD-ROM.
- Midterm Exam: 15%
- Final Exam: 25%

Exams

Before you finish Module 3, you will need to apply for your midterm exam. Before you finish Module 5, you will need to apply for your final exam. Here is how you apply for an exam:

- **If you are attending school**, ask your school's Independent Study Option (ISO) Facilitator to add your name to the ISO exam eligibility list. Do this at least three weeks prior to the next scheduled exam week.
- **If you are not attending school**, check the Examination Request Form for options available to you. The **Examination Request Form** was mailed to you with this course. Fill in this form and mail or fax it three weeks before you are ready to write the exam. The address is:

ISO Registration
555 Main St.
Winkler, MB R6W 1C4
Fax: 204-325-1717
Phone: 1-800-465-9915

How to Practise What You've Learned

It's really important to practise what you've learned. If you don't practise what you've learned, you won't have the confidence or experience when you are working on your assignments or writing your exam. We've included **Learning Activities** in each module to give you the chance to practise what you've learned.

Ask your study partner to keep the learning activity keys in a safe place until you have finished a given learning activity. After you have completed a learning activity, ask your partner to evaluate it by using the answer key. Repeat the process for all the learning activities in the course. The best way to learn is by being challenged. Use your highlighter pen to identify the questions which cause you difficulty and refer to these questions as you prepare to write the exams.

It is important to attempt to answer all the questions before looking at the answers. By copying the answers you will not learn nearly as much as if you struggle to find the answer yourself.

How Much Time Will You Need?

Learning computer science through independent study is a little different than learning it in the classroom. One of the advantages is that you are in charge of how you learn and can choose how quickly you will complete the course. You don't have to wait for your teacher or friends, and you can work as quickly as you want. You can also complete as many lessons at a time as you want. We really want you to succeed. Please read the next few pages to get an idea of how to pace yourself.

On the top of the first page of each lesson, write the date and time when you started and finished the lesson. In this way, you will have a record of the time that you spent on each lesson.

As the course progresses, the assignments become more involved and will take more time. A suggested amount of time for each module is:

- Module 1: Introduction to Computer Science (approximately 10 hours)
- Module 2: Introduction to Programming (approximately 20 hours)
- Module 3: Control Structures (approximately 25 hours)
- Module 4: Loops: Going in Circles! (approximately 25 hours)
- Module 5: Functions: Piecework Programming (approximately 30 hours)

Please note that these times are approximations. Depending on many factors, it may take you more or less time to complete the modules. As you see, you will be spending a minimum of 110 hours on this course. This includes time reading, studying, programming, completing assignments, and writing exams. That means at least 45 minutes daily to complete the course in a regular school year, or at least 90 minutes daily in a semester.

Take a look at the following three charts and decide which chart best describes the time of year that you want to finish the course. Share your study plans and timelines with your study partner, so that you know that you are on target with your goal of completing this course by a specific target date. *Failing to plan is planning to fail.* A very important step toward success is being able to plan and manage your time.

Chart A: Semester 1

Here is a suggested timeline that you can follow if you have registered for this course in September and plan to complete it by January.

Module 1	Assignments 1.1, 1.2, and 1.3	September 20
Module 2	Assignments 2.1, 2.2, and 2.3	October 10
Module 3	Assignments 3.1, 3.2, and 3.3	October 25
	Midterm Exam	October 31
Module 4	Assignments 4.1, 4.2, and 4.3	November 20
Module 5	Assignments 5.1, 5.2, and 5.3	December 15
	Final Exam	January 10

Chart B: Semester 2

Here is a suggested timeline that you can follow if you have registered for this course in January and plan to complete it by June.

Module 1	Assignments 1.1, 1.2, and 1.3	February 15
Module 2	Assignments 2.1, 2.2, and 2.3	March 5
Module 3	Assignments 3.1, 3.2, and 3.3	March 20
	Midterm Exam	April 5
Module 4	Assignments 4.1, 4.2, and 4.3	April 30
Module 5	Assignments 5.1, 5.2, and 5.3	May 25
	Final Exam	June 5

Chart C: Full School Year (not semestered)

Here is a suggested timeline that you can follow if you have registered for this course in September and plan to complete it by June.

Module 1	Assignments 1.1, 1.2, and 1.3	September 20
Module 2	Assignments 2.1, 2.2, and 2.3	October 20
Module 3	Assignments 3.1, 3.2, and 3.3	December 15
	Midterm Exam	January 10
Module 4	Assignments 4.1, 4.2, and 4.3	March 25
Module 5	Assignments 5.1, 5.2, and 5.3	May 25
	Final Exam	June 5

Do not wait until the last minute to complete your work, since your tutor/marker may not be available to mark it immediately. Remember, it might take over a week for your work to travel through the mail. It may also take a few weeks for your tutor/marker to mark everything and send your marks to your school.

When Do You Mail Things In?

You'll be emailing or mailing something to your tutor/marker at the end of each of the five modules. Here is a chart showing what you will be sending in at the end of each module.

Module	Items You'll Be Sending In
Module 1	Assignments 1.1, 1.2, and 1.3
Module 2	Assignments 2.1, 2.2, and 2.3
Module 3	Assignments 3.1, 3.2, and 3.3
Module 4	Assignments 4.1, 4.2, and 4.3
Module 5	Assignments 5.1, 5.2, and 5.3

How Do You Send in Your Assignments?

In this course, you have the choice of either mailing or emailing your assignments.

- Each time that you mail something, you must include the print version of the applicable Cover Sheet (found at the end of this Introduction).
- Each time that you email something, you must include the electronic version of the applicable Cover Sheet (found at <www.edu.gov.mb.ca/k12/dl/downloads/index.html>).

Complete the information at the top of the Cover Sheet before mailing or emailing it along with your assignments.

Mailing Your Assignments

If you choose to mail your completed assignments, please photocopy all of the materials first so that you will have a copy in case your package goes missing. You will need to place the applicable module Cover Sheet and assignments in an envelope and address it to

ISO Tutor/Marker
555 Main Street
Winkler MB R6W 1C4

Your tutor/marker will mark your work and return it to you by mail.

Emailing Your Assignments

If you choose to email your assignments, make sure you have saved copies of them before you send them. That way, you can refer to your assignments when you discuss them with your tutor/marker.

To email your completed assignments, you will first need to do one of the following:

- **If you are attending school**, please ask your ISO school facilitator (the person who signed your ISO Registration/ Admission Form) for permission to email your assignments and to determine your school's procedure for emailing assignments/unsupervised tests. Contact your tutor/marker to confirm that the course material can be marked electronically.
- **If you are not attending school**, please obtain permission directly from your tutor/marker to submit your assignments electronically.

How to Submit Your Work (files must not exceed 5 MB)

Please submit your work in the file types shown below:

- Written work: Microsoft Word files (doc) or RTF files
- Spreadsheets: Microsoft Excel files (xls)
- Pictures and graphics: JPEG or GIF files
- Scanned work: PDF files (save multiple pages on one file)
- Audio recordings: WAV files
- Video recordings: WMV files

How to Send Your Email

1. Use the following format to compose your email.

To:	distance.learning@gov.mb.ca
cc:	[your ISO school facilitator's email address if you attend school]
Subject:	[My Name] Grade 10 Computer Science
Attachment:	Assignment 1.1.doc
Message:	Assignment 1.1
	Tutor/marker: _____
	School: _____

2. Attach your files (files must not exceed 5 MB).
3. Email your assignments to <distance.learning@gov.mb.ca> only. Do **not** email your assignments directly to your tutor/marker. Emails sent directly to your tutor/marker will be returned unread.

Your tutor/marker will mark your work and return it to you by email.

Guide Graphics

Guide graphics are included throughout the course to help you identify specific tasks you need to complete. Here is an explanation for each of them:



Learning Activity: Complete this to practise what you have learned.



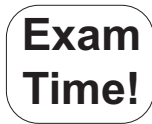
Mail-In Assignment: This icon indicates there is an assignment you will complete for marks that needs to be mailed or emailed to your tutor/marker.



Note: This icon signifies important and significant material.



Outcomes: This icon indicates what you will learn.



Exam Time: This icon reminds you that it is time to make arrangements to write your Final Examination.

GRADE 10
COMPUTER SCIENCE (20S)

Module 1
Introduction to
Computer Science

Module 1

Introduction to Computer Science



Module 1 Overview

In this module you will explore the evolution of computer hardware looking at the history of computers as it relates to computer science. You will also learn about computer ethics and careers in computer science. You will start programming in Module 2.

This module includes one self-assessed learning activities. These activities are for you to think about what you have read. Some of the learning activities have answers that can be found in the text of the lesson. None of the learning activities need to be handed in. There are three assignments to be submitted for marking. By the end of this module, you will:

1. Develop a general understanding of the essential components of a computer.
2. Perform an independent analysis of a specific topic relating to the history of computing.
3. Discuss the implications of the progressive development of computer hardware for the environment and for society.
4. Investigate the question of ethics in computing technology looking at computer crime and security practices.
5. Explore career opportunities in information and communication technologies that relate to computer science.

Notes



Lesson 1

Language and Machines

Why can't we just tell a computer what it needs to do in normal, everyday English? Why do we have to use a computer language when we write a program? In order to answer these questions, we need to understand a little bit of what has come before, that is, a little of the history of language and machines. We'll see how these two concepts, when put together, led to the creation of modern digital computers and programming languages.

In this first lesson, we look at early calculating devices, the first attempts at using the language of mathematics on machines.

By the end of this lesson, you will be better able to:

- define “algorithm”



Algebra and Algorithms Explained

Algebra is a formal language of symbol and substitution. It is an abstract construct that allows us to create general rules for handling whole sets of mathematical problems. Early mathematicians were able to use the formal language of algebra to create algorithms. These algorithms allowed them to solve problems not specific to particular numbers.

In the past, algorithms were used mostly in algebra. Algorithms have a much broader meaning today; they are the set of step-by-step logical instructions needed to solve a stated problem.

You do not need to be a rigorous mathematician to be a good programmer, but understanding how a formal language like mathematics developed and how it is applied to the real world gives us insight into how and why programming languages have come into being. If we are given a way to express problems—and the path to their solution—in an exact and methodical way, the calculation of the actual solution is essentially mechanical. The mechanics can be left up to a machine.

The Calculating Machine

At the time of the European Renaissance which started in the early 1500s, there were generally accepted methods for calculating relationships between numbers. These methods could be applied to activities in the real world. The use of machinery in agriculture and other fields was proving that some tasks could be accomplished by the use of human-made tools. It would only be a matter of time before people started to look to machines to replace the mundane step-by-step process of calculation.

Human Calculators

The rebirth of the study of mathematics in Europe and the ability to publish and share the fruits of this study among many thinkers led to a formal language of computation. Given a problem, a mathematician in England could devise an algorithm and effectively describe his solution using the language of mathematics. A group of human calculators in Italy could then apply his solution to a set of data and tabulate the results for an engineer to build a bridge or for an artillery officer to use in the field.

Because the mathematician in England and the group of “human calculators” in Italy had a common formal language to describe and interpret mathematical problems, they were able to apply these formulas to solve problems in the real world.

However, there was a problem. Human calculators have a tendency to get bored, sleepy, and inattentive to the mundane task of calculation. Given a way to find a solution, no creativity or insight was needed to produce a set of results from a set of data. Thinkers began to realize that a machine, not subject to the boredom of repetitive tasks, would not make the sloppy errors that human calculators were so prone to.

The initial challenge faced by inventors was how to get a machine to perform the basic fundamental operations of arithmetic: addition, subtraction, multiplication, and division. If complex mathematics had been built upon these basic operations, and if a machine could perform these basic operations, perhaps more complicated mathematical functions could be accomplished by a machine in the future.

Calculating Devices

In 1642, the French mathematician and philosopher Blaise Pascal created a machine called the Pascaline, to help his father, a tax collector, with the laborious calculations associated with his job. The Pascaline was a numerical wheel calculator that had eight movable dials to add sums up to eight figures long. The machine used a base of ten for calculation. When the first dial, representing 0 to 9 was moved ten notches, it incremented the next dial by one, representing the tens column. This dial, upon reaching ten, in turn incremented the next dial, representing the hundreds column one notch. The limit of the Pascaline was that it could only perform addition and only with numbers of eight figures.

In 1694, Gottfried Leibnitz, a German mathematician, demonstrated his Stepped Reckoner a considerable improvement over the Pascaline. This machine, instead of using the flat gear method of the Pascaline, had a movable stepped-drum carriage that was able to do multiplication by repeated automated additions. It was able to work on operands of up to 5 and 12 digits and display a product of up to 16 digits. Limitations of the Stepped Reckoner were that in order to use the carry mechanism, the user had to intervene during the operation of the machine, and because of the lack of precision parts, its results were not always reliable.

Charles Babbage

During the 19th century, a number of people improved calculating machines. One of these was an eccentric English professor of mathematics and inventor named Charles Babbage. Babbage was frustrated with the amount of time and errors involved in the compilation of logarithmic tables by human calculators. He believed that machines were able to perform repeated tasks without mistake. He also realized that, once a machine was able to perform one calculation, it was relatively easy to have it repeat that calculation over and over.

In 1822, he created a design for a machine called the Difference Engine. This machine would be able to compute (as in computer) differential equations by using gears and rods. Remember, at that time, machines had to run without electricity. He won an award and was given a large grant to construct a model of his design.

In 1832, he built a prototype of the Difference Engine that could operate on six-digit numbers and second-order differences (able to tabulate quadratic polynomials). In other words, it was one of the first machines that could do more than add, subtract, multiply, and divide. Regretfully, the prototype never developed into a full-blown Difference Engine. Babbage spent the rest of his life working on a new model called the Analytical Engine.

The Analytical Engine was the first documented attempt at building a machine we now know as a modern computer. It was designed to be a general purpose calculator that could perform the four basic arithmetic functions and could also do higher mathematics through a set of instructions that could be programmed into the machine by programmers using cards with holes punched in them. It was also able to remember a number that could be used in later calculations.

The Analytical Engine was never completely built for several reasons. One of the main ones was that it was impossible to build parts as precise as the ones needed. However, Babbage was recognized as the grandfather of the modern computer because of his vision and his designs.

Ada Lovelace

Ada Lovelace was the teenage daughter of the English poet Lord Byron. She had a passion for mathematics and was one of the few people who understood what Babbage was trying to do. She had a lifelong relationship with him and helped him with the Analytical Engine. Ada has been called the first programmer because she was the first to understand how a machine could perform a set of instructions using a program based on punch cards. It was Ada who first conceived of conditional jumps, loops, and subroutines.

Ada experimented with writing sequences of instructions. She noticed that, in order to perform complex calculations, the subcalculations had to be repeated many times. This was very tedious to do by hand so she proposed that they should be stored and then retrieved mechanically. These are now called subroutines and are used in programming today.

Ada also invented an instruction that would make the machine refer to a previous punch card. This way, the instruction could be repeated a number of times. This is called a 'loop' in modern programming.

Ada also conceived of the 'if' statement. This procedure allowed the machine to jump to another card in any part of the sequence 'if' a specific condition was satisfied. The 'if' statement meant that the machine could do more than just calculate; it could make decisions.

It wasn't until 1820 that a Frenchman, Thomas de Colmar, invented a machine called the Arithometer. It was capable of the four basic arithmetic functions and was based on the similar gear and drum design of Leibnitz's Stepped Reckoner. The Arithometer was better designed and used superior parts that could only have been made using the machine-making tools of the 19th century. Colmar's machine, and machines like it, were used and manufactured up until World War I.

Reaching the Limit

The invention of calculating machines enabled people to leave the mundane task of adding, subtracting, multiplying, and dividing large numbers to a machine. But these machines could only handle simple math. The size and speed of the calculations they could perform limited their use.

Calculating machines continued to be designed and built into the 20th century. One of the difficulties of building a machine capable of replicating the four fundamental operations of arithmetic was that a complicated series of gears and dials, rods, or drums were necessary to perform the calculations. It wasn't until the 19th century that machine makers were able to create the parts needed with enough precision and hardness to withstand repetitive and constant use.

Calculating the Future

Calculating devices offered the world a new way of applying technology to calculation. Their widespread use in the 19th and early 20th centuries replaced human calculators in the workplace. But, they were incapable of performing complex mathematics or operations on very large numbers in a reasonable time. In the pursuit of a device capable of higher math and quick reliable calculations on very large numbers, a new vision of the machine and a new vision of mathematics for the machine had to be developed. It would become a way to build upon the calculating machine, not unlike the way mathematics built itself from arithmetic. This vision would start in the 19th century but not be fully realized until the mid-20th century.

Notes



Released 2010



Printed in Canada
Imprimé au Canada