

MAKERERE UNIVERSITY
FACULTY OF COMPUTING AND
INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Computer Science Subject
for BSc program

1 Introduction

1.1 The Bachelor of Science (B.Sc) Program

The B.Sc program is housed in the faculty of Science. It is a broad program that explores a wide range of knowledge in Natural Sciences and other related disciplines. The program is subject based. Some of the subjects like Botany, Mathematics, Physics, Zoology, and Biochemistry are housed in the faculty its self. However, some subjects like Computer Science, Geography, Statistics and Economics are housed in other faculties.

The faculty of Computing and Information Technology houses the Computer Science subject. The department of Computer Science is responsible over the day today running of the subject. All administrative and academic decisions on the subject are done in consultation with the Faculty of Science.

1.2 The Computer Science Subject

It has been an aspiration of Makerere University to be a mover in the information age. This is done by training highly skilled computer professionals. As a starting point, Computer Science as a subject was started in the Faculty of Science in the early 1990's. Its teaching was later transfered to the Institute of Computer Science (now faculty of Computing and Information Technology). The subject is offered by students on the Bachelor of Science program and is done with other subjects in the Faculty of Science.

In the Faculty of Computing and Information Technology, the department of Computer Science is responsible over the day today running of the subject. This includes teaching as well as devising ways the contents can be enriched further to match the current ICT trends.

Basing on the increased demand for Computer professionals, the Department of Computer Science in Consultation with the Faculty of Science seeks to revise the curriculum for the Computer Science subject. Overall, the revision is aimed at improving the skills the students acquire so as to make them more suitable for the ICT work environment. Specifically, the revisions seek to

- Increase the practical orientation of the subject
- Increase the theoretical foundations of the subject
- Ensure that all courses taught contribute bigger solid knowledge areas necessary for a practicing computer scientist

1.3 Major changes in the curriculum

The major changes in the curriculum are:

1. Integrating some course units:
In the old curriculum, many course units were done in phases, this has been removed by integrating them into a single course unit that cover the entire material.
2. Increasing the practical component:
There are more hours allocated for practical. In some cases, some of the courses have a practical component of the final examination.
3. Increasing the theoretical component:
More courses on the theoretical fundamentals of computer science have been introduced

1.4 Knowledge areas in the new Curriculum

The key knowledge areas in the curriculum are:

- Theoretical Computing
- Data structures, Algorithms and Programming
- Organizational Aspects of Software Development
- Research and Development

2 Regulations

2.1 Subject Combinations

Computer science subject forms part of the Bachelor of Science program. It has to be done with two other subjects at enrollment. Due to the nature of Computer Science, it shares a lot with Mathematics. Therefore a candidate have to demonstrate a strong mathematics background (for example good grades from A-Level) to be offered the subject.

2.2 Admission Requirements

A student has to satisfy the general university admission requirements as well as those specific ones on the BSc program. Particularly for Computer Science, a student must have a principle pass in Mathematics at A' level.

2.3 Assessment

Assessment is to be done by progressive assessments (like tests, assignments, group work) during the semester and final examination. The final examination may be purely written, purely practical or having a written and practical component. Progressive assessments constitute 40% of the final score and the final examination will constitute 60%.

2.4 Grading and Pass mark

Grading will be based on the final score for each examination using the ranges below

Final	Letter Grade	Grade Point
90-100	A+	5
80- 89	A	5
75- 79	B+	4.5
70- 74	B	4
65- 69	C+	3.5
60- 64	C	3
55- 59	D+	2.5
50- 54	D	2
45- 49	E+	1.5
40- 44	E-	1
0 - 39	F	0

A student with a grade point greater or equal to 2 (letter grade D) in a certain course unit is considered to have passed the course unit.

2.5 Graduation Load

Computer Science forms a portion of the overall load on the BSc program. The minimum graduation load for B.Sc is 108 Credit units. Consultation with Faculty of Science showed that a majoring student needs between 66 and 72 credit units while a minoring student needs between 30 and 36 CU for graduation.

2.6 Computer Science Major

Code	Name	CU	LH	PH	TH	CH
Semester I						
CSC 1100	Computer Literacy	4	30	60	–	60
CSC 1106	Programming Methodology I	3	30	30	–	45
	Total	7				
Semester II						
CSC 1207	Programming Methodology II	4	30	60	–	45
CSC 1209	Logic Programming	3	30	30	–	45
	Total	7				
Semester III						
CSC 2100	Algorithms and Data Structures	4	45	–	30	60
CSC 2111	Database Management Systems I	3	30	30	–	45
CSC 2113	Software Engineering	4	45	–	30	60
	Total	11				
Semester IV (Core)						
CSC 2200	Operating Systems	4	45	–	30	60
Semester IV (2 electives)						
CSC 2209	Systems Programming	4	45	–	30	60
CSC 2210	Automata, Complexity & Computability	3	45	–	–	45
CSC 2214	Cryptology and Coding Theory	3	45	–	–	45
	Total	10				
Semester V (Core)						
CSC 3103	User Interface Design	4	45	–	30	60
Semester V (2 electives)						
CSC 3112	Principles of Programming Languages	3	45	–	–	45
BIS 3100	Modeling and Simulation	4	45	–	30	60
CSC 3105	Computer Graphics	3	30	–	30	45
CSC 3115	Advanced Programming	3	30	–	30	45
	Total	10				
Semester VI (Core)						
CSC 3206	Group Project	5	–	135	–	60
Semester VI (2 electives)						
CSC 3205	Compiler Design	3	45	–	–	45
CSC 3207	Computer Security	3	45	–	–	45
BIT 3200	Business Intelligence & Data Warehousing	4	45	30	–	60
BSE 3202	Distributed Systems Development	4	45	30	–	60
	Total	11				

2.7 Computer Science Minor

Code	Name	CU	LH	PH	TH	CH
Semester I						
CSC 1100	Computer Literacy	4	30	60	–	60
CSC 1106	Programming Methodology I	3	30	30	–	45
	Total	7				
Semester II						
CSC 1207	Programming Methodology II	4	30	60	–	45
CSC 1209	Logic Programming	3	30	30	–	45
	Total	7				
Semester III						
CSC 2111	Database Management Systems I	3	30	30	–	45
CSC 2113	Software Engineering	4	45	–	30	60
	Total	7				
Semester IV (Core)						
CSC 2200	Operating Systems	4	45	–	30	60
Semester IV (1 elective)						
CSC 2210	Automata, Complexity & Computability	3	45	–	–	45
CSC 2214	Cryptology and Coding Theory	3	45	–	–	45
	Total	7				
Semester V (Core)						
CSC 3103	User Interface Design	4	45	–	30	60
Semester V (1 elective)						
CSC 3112	Principles of Programming Languages	3	45	–	–	45
CSC 3105	Computer Graphics	3	30	–	30	45
	Total	7				
Semester VI (Any 2)						
CSC 3205	Compiler Design	3	45	–	–	45
CSC 3207	Computer Security	3	45	–	–	45
BIT 3200	Business Intelligence & Data Warehousing	4	45	30	–	60
BSE 3202	Distributed Systems Development	4	45	30	–	60
	Total	6				

3 Course Details

3.1 Semester 1

3.1.1 CSC 1100: Computer Literacy

(a) Description

In this course, students are to learn about the basic organization, concepts and terminologies in a computerized environment. They are also to get an in depth understanding of common computer applications. The use of related applications in different operating systems will be explored.

(b) Aims

The aims of the course unit are to:

- Equip students with basic knowledge about computer organization;
- Equip students with skills of using common office applications;
- Expose students to different operating systems;
- Equip students with skills of how to use the Internet and
- Equip students with knowledge about common text editors in different operating systems.

(c) Learning outcomes

By the end of the course unit, the student should be able to:

- Describe the different parts of a computer;
- Describe the historical evolution of computers;
- Competently use the common office applications in at least two operating systems;
- Competently use common text editors in at least two operating systems.

(d) Teaching and learning pattern

Teaching will be by lectures and laboratory demonstrations/practicals

(e) Indicative content

- General computer organization
- Historical perspectives of computing

- Common Microsoft office packages
 - Office packages in other operating systems
 - Text editors
 - Common Linux commands
 - Using the web
- (f) Assessment method
The assessment will be in form of tests and assignments (40%) and final written exam (60%)
- (g) Reading list
- (i) Computer Literacy by John Preston, Robert Ferrett and Shelly Gaskin, 2007.
 - (ii) Practical Computer Literacy by Jelne Janrich and Dan Oja, 2001

3.1.2 CSC 1106: Programming Methodology I

- (a) Description
The course is to create a strong base in the principles and practice of functional programming. A high level programming language like C is to be used. Students are to cover both theoretical principles and hands on practical skills. The main concepts to cover include program structure, data structures, syntactical and semantic correctness, planning and segmentation in programming as well as working with files.
- (b) Aims
The aims of the course are to provide the student with:
- Comprehensive knowledge about structured oriented programming;
 - Knowledge in planning and organization of programming projects;
 - Knowledge and techniques of evaluating syntactic and semantic correctness of a computer program;
 - Strong practical basis in programming.
- (c) Teaching and Learning pattern
The course will be taught with a big practical component. Students will be expected to have one supervised practical sessions per week. They will so be given several programming assignments some of which will be marked and contribute to the coursework scores.

(d) Indicative content

- Program structure
- Variables and Operators
- Conditional statements
- Looping statements
- Arrays and strings
- Functions
- Advanced data types
- Pointers
- Dynamic memory allocation and dynamic structures
- Working with files
- GUI

(e) Assessment method

Assessment will be in form of at least one (practical) assignment and one test (40%), A practical exam - (30%) and a final written examination (30%)

(f) Reading list

- (i) C Programming Language by Brian W. Kernighan, Dennis M. Ritchie; Prentice Hall, 2000.
- (ii) C: A Reference Manual (5th Edition) by Samuel P. Harbison; Prentice Hall, 2002.

3.2 Semester 2

3.2.1 CSC 1207: Programming Methodology II

(a) Description

The course is to give an in depth understanding of Object Oriented programming. It is to cater for Object Oriented programming practices like inheritance, interfaces, exception handling, action handling, security, software reuse and robustness.

(b) Aims

The aim of the course is to

- Move the students' programming skills from basic to advanced

- Avail students with skills to handle non functional program aspects like robustness and security
 - Train students to develop complete computer applications
- (c) Teaching and learning pattern
This will include lectures, practicals and lab assignments
- (d) Indicative content
- The object oriented paradigm
 - Classes and objects
 - Inheritance and visibility modifiers
 - Interfaces and abstract classes
 - Graphical user interface and action handlers
 - Exception handling
 - Working with files
 - Working with databases
 - Sessions and user management
- (e) Assessment method
The assessment will be done by tests and take home assignments (40%), practical examination (30%) and written examination (30%)
- (f) Reading list
- (i) Java Software Solution: Foundations of Program Design (6th Edition) by John Lewis and William Loftus, Addison-Wesley, 2009.
 - (ii) A Programmer's Guide to JavaTM Certification: A comprehensive Primer by Khalid A. Mughal and Rolf W. Rasmussen, Addison-Wesley, 1999.

3.2.2 CSC 1209 Logic Programming

- (a) Description
This course introduces a paradigm where computation arises from proof search in a logic according to a fixed, predictable strategy. It thereby unifies logical specification and implementation in a way that is quite different from functional or imperative programming. This course provides a thorough, modern introduction to logic programming. It introduces the basic concepts and techniques of logic programming followed

by successive refinement towards more efficient implementations or extensions to richer logical concepts. It covers a variety of logics and operational interpretations.

(b) Aims

The aim of the course is to provide a basic introduction to the logic programming language, Prolog. It aims at introducing a number of logical systems of importance in computer science.

(c) Learning outcomes

By the end of the subject, students should:

- Be conversant with the syntax and semantics of propositional and predicate logic
- Be familiar with a variety of applications of predicate logic in software verification, databases and knowledge-based systems
- Be able to write specifications in predicate logic expressing state constraints
- Understand the notion of formal proof, and be able to construct simple proofs in a natural deduction proof system for predicate logic
- Be aware that there are inherent expressiveness and computational limitations to the applicability of logical systems, and be familiar with a number of restrictions under which the computational limitations can be overcome
- Be able to write programs in a logic programming language,
- Understand both the top-down and the bottom up operational semantics of logic programs
- Be familiar with a logic for reasoning about sequential programs, and capable of constructing correctness proofs for simple programs

(d) Teaching and Learning pattern

The course consists of a traditional lecture component and a project component. The lecture component introduces the basic concepts and techniques of logic programming. The project component will be one or several projects related to logic programming.

(e) Indicative content

- Introduction

- Pure logic (relational) programming
 - The Prolog Language
 - Programming in Prolog.
 - Efficient Prolog Programming
 - Combining Logic Programming, Functional Programming, Higher Order, Objects.
 - Review of first order predicate logic and resolution.
 - Fundamental results.
 - Semantics of logic programs.
 - Implementation of logic languages and advanced compilation.
 - Parallelism, concurrency.
 - Other LP/CLP languages
- (f) Assessment method
Assessment will be by assignments and/or tests (40%) and written examination (60%)
- (g) Reading list
- (i) Logic in Computer Science, Modeling and Reasoning about Systems, M.R. Huth and M.D. Ryan, Cambridge University Press 2000
 - (ii) SWI Prolog Home Page, <http://www.swi-prolog.org/>

3.3 Semester 3

3.3.1 CSC 2100: Data Structures and Algorithms

- (a) Description
The course gives students a firm foundation of data structures and algorithms. The course trains students on systematic development and analysis of algorithms. The importance of algorithm complexity on computer performance is emphasized. Typical computational problems and their solutions/analysis are to be covered.
- (b) Aims
The aims of the course are to
- Make students appreciate the role of data structures and algorithms in computer programs;

- Improve students' problem solving skills by subjecting them to step by step analysis and design of computer algorithms;
 - Introduce students to concepts Data structures;
 - Introduce students to concepts of algorithm analysis;
 - To expose students generic algorithmic problems and apply them to other computational scenarios.
- (c) Teaching and Learning pattern
The teaching pattern is by lecture, practical lab work, group discussion and class presentations.
- (d) Indicative content
- Complexity analysis (Big-O notation, orders of growth, worst case, average case and amortized analysis);
 - NP-complete problems;
 - Greedy algorithms;
 - Dynamic programming;
 - Design patterns for data structures;
 - Graphical representation of optimization problems
 - Parallel algorithms.
 - Sorting and searching;
 - Divide-and-conquer algorithms;
 - Elementary data structures;
 - Recursive data structures (stacks, queues, linked lists, trees);
 - Storing and searching (hash tables, search trees);
 - Graph algorithms on graphs (shortest path, spanning trees).
- (e) Assessment method
The assessment will be done by tests/assignment (40%) and final examination (60%)
- (f) Reading List
- (i) Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, John E. Hopcroft. Addison-Wesley, 1983
 - (ii) The Design and Analysis of Computer Algorithms by Alfred V. Aho, Addison-Wesley Longman, 1974

- (iii) Introduction to Algorithms 2nd Ed by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, McGraw-Hill, 2008

3.3.2 CSC 2111: Database Management Systems I

(a) Description

The course is provide students with a strong foundation in systematic approaches to design and implementation of database applications. Preliminary operations like requirements gathering and database planning will be covered. The course will also introduce students to developing of application programs that talk to the database. These applications may be online or off line.

(b) Aims

The aims of the course are to:

- Provide a background for the evolution of database (management) systems
- Provide the students with the steps one has to go through when developing good database applications
- Give hand on experience and knowledge in developing database (driven) applications

(c) Teaching and Learning patterns

Teaching will be by lectures, take home reading assignments/class presentations and laboratory practicals

(d) Indicative content

- Background to databases
- Evolution of database systems
- Database organization and architecture
- Database models
- The database development life cycle
- Database design
- Tuning of operational systems
- Querying databases
- SQL/PL SQL

- Scripting
- (e) Assessment Methods
Assessment will be in terms of tests and take home assignments (40%),
A practical examination (30%) and a final written exam (30%)
- (f) Reading list
- (i) Thomas Connolly and Carolyn Begg: Database Systems: A Practical Approach to Design, Implementation and Management. 2nd Edition, Addison-Wesley, 2004.

3.3.3 CSC 2113: Software Engineering

- (a) Description
This course introduces students to the foundations of software engineering as a discipline. Students are introduced to the evolving role of software engineering, especially with emphasis on software engineering process and process models. Key topics covered include Software configuration management, Requirement analysis, Software Specification, Design methods, Software testing, Software project management techniques; Software project planning, Risk management; Software Quality Assurance; Software reuse; and Computer aided software engineering: CASE tools and application.
- (b) Aims and Objectives
- To introduce software engineering and to explain its importance
 - To set out the answers to key questions about software engineering
 - To introduce ethical and professional issues and to explain why they are of concern to software engineers
- (c) Learning outcomes
On successfully completing of this unit students will be able to
- Demonstrate competence in handling software engineering projects.
 - Know the fundamental software engineering processes and models
 - Know what is involved in a typical software engineering project's life cycle.
 - Employ good project management principles in handling projects and know why these principles are important in constructing quality software.

- Be competent in using CASE tools in real world projects.
- (d) Teaching and Learning Pattern
- Teaching and learning is to be implemented through lecture, lab and tutorial sessions. Students are also expected to make presentations of their work.
- (e) Indicative Content
- Evolving role of software, software characteristics; Systems and environment; system engineering hierarchy, information and knowledge engineering; Information strategy; Business Area analysis, modeling enterprise and business-level data modeling, system architecture and associated information flow; writing system specification.
 - Software Engineering as a layered technology: Software process, software process models. Software configuration management: the SCM process, Identification of objects in software configuration, version control, change control, configuration audit, SCM standards.
 - Requirement analysis: Communication techniques, Information gathering tools; organizing and structuring information; analysis principles; Analysis modeling.
 - Software Specification: Design process, principles and concepts: Abstraction, refinement, modularity, control hierarchy, structural partitioning, information hiding, functional independence, cohesion, coupling, design heuristics;
 - Design methods: data design, architectural design, transform mapping, design optimization, human computer interface design, procedural design and tools; Design documentation.
 - Software testing: Testing objectives, Testing principles, Testability, test case designing, white box testing; Basis path testing: Condition testing, data flow testing, loop testing; Black box testing: graph based testing methods, equivalence partitioning, Boundary value analysis, comparison testing; Testing documentation and help facilities; Software testing strategy: unit testing, integration testing, validation testing, system testing.
 - Software project management techniques: project metrics, software measurement and metrics, software quality metrics;

- Software project planning: objectives of planning, resources, project estimation and estimation models, project decomposition techniques, make-buy decisions; automated estimation tools.
- Risk management: software risks, risk identification, risk projection, risk mitigation, monitoring and management; Project Scheduling: people and effort relationships, defining tasks, defining task network, scheduling techniques; Software teams and intra-team relationships; role of project manager.
- Software Quality Assurance: Concept of quality, quality control vs. quality assurance, cost of quality, factors that affect quality, quantitative view of quality, quality metrics, defect removal efficiency SQA activities, ISO standards and CMM practices, SEI levels, Software reviews, Formal approaches to SQA, Statistical Quality Assurance. Software reliability, reliability metrics, reliability models, meeting reliability requirements.
- Effective metrics for software process: Measurement principles, attributes of software metrics, metrics for analysis model, metrics for design model, metrics for source code, metrics for maintenance.
- Software reuse: difficulties in reuse, hardware reuse vs. software reuse, reusable artifacts, domain engineering approach, analysis design and construction of reusable components, classification and retrieval of components, economic impact of reuse and reuse metrics.
- Computer aided software engineering: CASE tools and application.

(f) Assessment method

- Continuous assessment through practical exercises and Coursework, together with two scheduled tests (40%)
- Final exam at the end of the Semester and accounts for 60% of the final grade.

(g) Reading lists

- (i) J.F. Peters, W.Pedrycz, Software engineering: An Engineering approach, John Wiley, 2000.
- (ii) R.S. Pressman, Software engineering: A Practitioners Approach, 5th Edition, McGraw Hill, 2005.

- (iii) Sommerville, Software engineering, 8th Edition, Addison Wesley, 2008.
- (iv) D. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of software Engineering, Prentice Hall of India, 2004.

3.4 Semester 4

3.4.1 CSC 2200: Operating Systems

(a) Description

Operating Systems course introduces students to software that controls hardware and makes the hardware usable. Its interaction with other computer devices and how it controls other computer processes is explored.

(b) Aims

The aims of the course are:

- To provide students with a detailed understanding of how operating systems work.
- To provide students with skills to write basic programs to utilize underlying operating system infrastructures.

(c) Learning outcomes

The dominant categories of operating systems are Windows and UNIX (Linux, Mac OS, Solaris, etc). Students of operating systems are expected to have a proper understanding of the differences between these two. Students should be able to understand different design principles for operating systems and various software tools that make operating systems usable.

(d) Teaching and Learning Pattern

The teaching pattern is by lectures, lab sessions and group projects.

(e) Indicative Content

- Operating Systems Structures
- Processes and threads
- Thread creation, manipulation and synchronization
- Deadlock
- Implementing Synchronization operations

- CPU scheduling
 - Memory management
 - File systems and file system implementation
 - Monitors
 - Segments
 - Disk Scheduling
 - Networking
 - UDP and TCP
- (f) Assessment method
The assessment will constitute Practical assignments on at least 5 chapters of the course and written course work (40%) and written Exam (60%)
- (g) Reading list
- (i) Operating Systems: Internals and Design Principles 5th Ed by William Stallings, Prentice Hall, 2005.

3.4.2 CSC 2209: Systems Programming

- (a) Description
Systems programming is aimed at teaching students how to write programs using system level services. The system of instruction is UNIX due to availability of free system tools that have been largely developed by and for the academia.
- (b) Aim
Skills in tools provided by systems, their commands, system calls and understanding for model of computation.
- (c) Teaching and Learning Pattern
The teaching pattern is by lectures, lab sessions and projects.
- (d) Indicative Content
- Introduction and Unix Standardization
 - File input and output
 - Standard I/O Library
 - Files and Directories

- System Data Files and Information
 - Process Environment
 - Process Control
 - Process Relationships
 - Signals
 - Threads
 - Advanced I/O
 - Interprocess Communication
- (e) Assessment method
Assessment will be in form of tests and practical assignment (40%) and final written examination (60%)
- (f) Reading List
- (i) Advanced programming in the Unix Environment, by W. Richard Stevens, Addison-Wesley 2008

3.4.3 CSC 2210 Automata, Complexity and Computability

- (a) Description
The course introduces students to the concept of automata and complexity. It sets a background for more advanced studies like compiler construction and principles of programming languages.
- (b) Aims The aims of the course are:
- To introduce students to the concepts of complexity, automata and computability
 - To prepare students for advanced studies in compiler construction and principle of programming languages
- (c) Teaching and learning pattern Teaching will be in form of class lectures and tutorials
- (d) Indicative content
- Finite state machines and regular languages:
 - Deterministic and non deterministic machines;
 - Equivalence and minimization;

- Regular expressions and regular grammars;
 - Kleene's theorem.
 - Push-down automata and context free grammars;
 - Normal forms of grammars; Top-down and bottom-up parsing.
 - Turing machines and computability;
 - Church's thesis;
 - NP-Computable problems.
- (e) Assessment method
Assessment will be in terms of Assignments and tests (40%) and final written examination (60%)
- (f) Reading list
- John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: Introduction to Automata Theory, Languages, and Computation; Addison Wesley, 2000.
 - Daniel I. A. Cohen Introduction to Computer Theory; John Wiley and Sons, Inc, 1996.

3.4.4 CSC 2214 Cryptology and Coding Theory

- (a) Description
This course provides a foundation for further studies in information security. The course introduces students to the exciting fields of cryptology and coding theory. Fundamentally, it deals with the mathematics that underlies modern cryptology. Cryptology combines the studies of cryptography, the creating of masked messages, and cryptanalysis, the unraveling of masked messages. Coding theory is the study of coding schemes used to detect and correct errors that occur during the data transmission.
- (b) Aims
The aims of the course are
- To understand the building blocks of crypto systems and error correction
 - To gain historical understanding of the evolution of crypto systems.
 - To develop tools necessary to crypto analyze crypto systems

- To gain insights in the practical application of cryptology and error correction in the modern information age.
- To understand the goals and trade-offs associated with encryption and error-control coding systems.

(c) Indicative Content

- History of cryptology and coding theory
- Shift registers
- Classical crypto-systems
- Stream ciphers
- Block ciphers
- Information theory
- Crypto analysis techniques
- Introduction to Elliptic curve cryptography
- Basic Algebra
- Coding theory fundamentals
- Linear codes
- Hamming codes
- Secret sharing schemes
- Introduction to Complexity
- Hash functions
- PGP & PKI Diffie-hellman key exchange protocol

(d) Learning outcomes

Upon successful completion of this course, the student should be able to

- Deploy sound cryptographic practices and tools
- Discuss the goals and trade-offs associated with encryption and error-control coding systems.

(d) Teaching and Learning Pattern

The course will be delivered in form of lectures, tutorials, and group assignments.

(f) Assessment method

At least 2 tests and 1 assignment (40%) One 3 hour examination (60%)

(g) Reading lists

- (i) Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996 Trappe & Washington,
- (ii) Introduction to Cryptography with Coding Theory, Prentice-Hall, 2001 ISBN 0130618144
- (iii) Introduction to Cryptography with Coding Theory by Trappe and Washington, 2nd edition, Prentice Hall, 2006.
- (iv) The Code Book by Simon Singh, Doubleday, 1999
- (v) Introduction to Algebraic Coding Theory with Gap, Sarah Spence Adams, 2005.

3.5 Semester 5

3.5.1 CSC 3103: User Interface Design

(a) Description:

The course introduces the principles of user interface development, focusing on design, implementation and evaluation.

(b) Aims

The course aims at providing the skills listed below to students:

- Developing efficient, flexible and interactive User Interfaces(UI)
- Provide ability to identifying system users, the tasks they want to carry out and the environment in which they will be working;
- Creating a conceptual designs;
- Designing various kinds of UI, in particular graphical user interfaces (GUIs) and web sites; evaluating UIs;
- Appreciation of realities of developing usable UIs in an organization.

(c) Teaching and Learning Pattern

The teaching pattern is by lectures, lab sessions and projects.

(d) Indicative content

- Usability
- User-Centered Design
- UI Software Architecture

- Human Capabilities
 - Output Models
 - Conceptual Models and Metaphors
 - Input Models
 - Design Principles
 - Paper Prototyping
 - Constraints and Layouts
 - Graphic Design
 - Computer Prototyping
 - Heuristic Evaluation
 - User Testing
 - Experiment Design
 - Experiment Analysis
- (e) Assessment method
Assessment will be in form of assignments and tests (40%), practical Exam (30%) and final written exam (30%)
- (f) Reading List
- (i) Norman, D. A. *The Design of Everyday Things*. New York, NY: Doubleday, 1990. ISBN: 0385267746.
 - (ii) Nielsen, J. *Usability Engineering*. Burlington, MA: Academic Press, 1994. ISBN: 0125184069.
 - (iii) Mullet, K., and D. Sano. *Designing Visual Interfaces: Communication oriented techniques*. Prentice Hall, 1994. ISBN: 0133033899.

3.5.2 CSC 3112: Principles of Programming Languages

- (a) Description
The course introduces students to the low level organization and operation of programming languages. It covers semantic and syntactic as well as operational issues in programming languages. The building blocks of programming languages are explored.
- (b) Aims
The aims of the course are

- To give students fundamental knowledge in the organization and operation of programming languages
- To make students appreciate the possible future evolutions of programming languages
- To expose students to causes of operational (like performance, security, etc) characteristics of programming languages

(c) Learning outcomes

By the end of the course, students will be able to:

- Understand common language paradigms
- Know the different building blocks of a programming language
- Know how the different blocks of a programming language interact

(d) Teaching and Learning Pattern

Teaching will be largely by lectures, tutorials and class assignments

(e) Indicative Content

- Overview over programming language paradigms
- Common principles: syntax, syntax trees, formal semantics (denotational and operational), variables and binding
- Types: role of types in programming and programming languages, types and their operations: products, sums, functions, recursive types, reference and array types
- Primarily imperative issues: control flow, arrays, pointers and references, parameter-passing mechanisms, scoping
- Type systems: strongly typed languages type checking (static vs. dynamic), type equivalence (by name vs. structural), overloading, coercion, polymorphism, type inference
- Binding: declarations and environments. Block structure: scope and visibility, stack discipline. Bound occurrences: static vs. dynamic binding.
- Encapsulation: information hiding, modules, abstract data types, classes
- Language implementation: parsing, code generation, garbage collection.

- (f) Assessment method
Assessment will be by Tests and Assignments (40%) and final written examination (60%)
- (g) Reading lists
 - (i) Friedman, Wand, and Haynes, Essentials of Programming Languages, MIT Press, 2001

3.5.3 BIS 3100: Modeling and Simulation

- (a) Description
The course gives students theoretical and practical skills in modeling and simulation of dynamic systems with a view of learning their behavior and the sensitivity of that behavior to certain parameters.
- (b) Aims
The aims of the course are:
 - Familiarize students with modeling and simulation techniques that are applicable under varying circumstances
 - Equip students with practical experiences of composing models and running simulations under varying circumstances
 - Equip students with skills of correctly representing simulation results
- (c) Teaching and Learning Method
Teaching and Learning will be in form of Lectures and laboratory demonstrations
- (d) Indicative Content
 - Simulation of operational systems
 - Simulation as a decision making methodology
 - Model development and validation,
 - Design of simulation experiments,
 - Generation of appropriate values of random variables,
 - Interactive procedures and interpretation of results.
- (e) Assessment Method Assessment will be in form of tests and assignments (40%) and final examination (60%)

(f) Reading List

- (i) Business Modeling and Simulation by Les Oakshott, 1997, Trans-Atlantic Publications
- (ii) System dynamics modeling a practical approach by R.G. Coyle, Chapman & Hall/CRC, 1996.

3.5.4 CSC 3105 Computer Graphics

(a) Description

The course covers general purpose graphics systems and their use. It gives an in depth knowledge of computer graphics and graphical user interfaces.

(b) Aims

The aims of the course are:

- Introduce students to the concepts of graphical representation on computers
- Teach students the design of good graphical user interfaces

(c) Teaching and Learning Pattern

Teaching will be in terms of class lectures and tutorials

(d) Indicative Content

- Graphics hardware,
- Geometrical transformations,
- Surface and volume visualization,
- Design and implementation of graphical user interfaces.
- Two dimensional imaging processes.
- Computer graphics applications.
- Display system organization;
- Display devices and modes;
- Display file construction and its structure;
- Graphic primitive - device initialization,
- view porting and windowing;
- Line drawing,

- simple and symmetrical Digital Differential Analysis (DDA);
 - Arch and circle generating DDA Line; and polygon clipping algorithms;
 - Curve plotting;
 - Transformations- projections and perspective views;
 - Picture segmentation: Graphics standards - PHIGS and GKS.
- (e) Assessment method Assessment will be in terms of assignments and tests (40%) and final exam (60%)
- (h) Reading lists
- (i) Introduction to Computer Graphics by James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes and Richard L. Phillips, Addison Wesley, 2003.
 - (ii) Fundamentals of Computer Graphics by Peter Shirley. AK Peters, 2002.

3.5.5 CSC 3115 Advanced Programming

- (a) Description
This course highlights programming practices that are vital in the day today work of a programming professional. While many systems are described by functionalities, some important aspects like security, robustness, and maintainability are ignored. Students are to get an in depth understanding of these concepts as well as exploring the current trends in the programming environment.
- (b) Aims
The aim of the course is to concretize the student's past programming experience as well as highlighting critical practices in programming that are necessary for a professional programmer
- (c) Learning outcomes
By the end of the course, the student should be:
- Able to implement non functional but critical aspects of programming like robustness and security
 - Able to develop well documented and well structured software that can easily be maintained

- Knowledgeable in other programming practices like mobile programming
 - Aware of newer programming paradigms like service oriented and cloud computing
- (e) Teaching and Learning Pattern
Teaching will be by lectures and lab demonstrations.
- (f) Indicative Content
- Programming for Security:
 - Programming for Robustness
 - Programming for Maintainability
 - Trends in Programming Paradigms
- (g) Assessment method
Assessment will be by tests and practical assignments (40%) and final written examination (60%)
- (h) Reading lists
- (i) Advanced Programming in the UNIX Environment by W. Richard Stevens and Stephen A. Rago Addison WEsley 1992

3.6 Semester 6

3.6.1 CSC 3205 Compiler Design

- (a) Description
In this course unit, students shall understand the complete process of translating a program in a high-level language to machine language. The course gives an introduction to the design and implementation of a compiler with emphasis on principles and techniques for program analysis and translation. It also gives an overview of the tools for compiler construction. Lexical analysis, token selection, transition diagrams, and finite automata. The use of context-free grammars to describe syntax, derivations of parse trees, and construction of parsers. Syntax-directed translation schemes; Intermediate code; Symbol table; Code generation; Detection, reporting, recovery and correction of errors.
- (b) Aims
The aim of the course is to allow students to examine how a high-level

language program is accepted as input and translated into assembly language or machine language so that the central processing unit receives instructions which it understands and can execute.

(c) Teaching and Learning pattern

The course consists of a traditional theoretical component and a project component. The lecture component introduces the basic concepts of compiler writing. The project component will involve students in writing a compiler for a specified programming language.

(d) Indicative content

- Language translators: Introduction to compilers and interpreters.
- The structure of a compiler: lexical analysis, parsing, semantic analysis.
- Intermediate code generation, register allocation, global optimization.
- Lexical scanning
- Parsing
- Automatic parser construction. FIRST and FOLLOW functions. LL(1) parsers. LR parsers. Conflicts in LR grammars and how to resolve them
- Semantic analysis: Attributes and their computation, tree-traversals, visibility and name resolution. Inherited attributes and symbol tables. Name resolution in block-structured languages
- Type checking: Type systems, varieties of strong typing, overload resolution, polymorphism and dynamic dispatching. Type-checking and type inference, unification
- Run-time or Run-time organization: storage allocation, non-local references, parameter passing, dynamic storage allocation. Exception handling, debugging information
- Intermediate code generation: control structures, expressions, simple register allocation. Aggregates and other high-level constructs
- Global optimization

(e) Assessment method

Assessment will be by assignments and/or tests (40%) and written examination (60%)

(f) Reading list

- (i) Compiler construction by William McCastline Waite, Gerhard Goos Springer Verlag 1994

3.6.2 CSC 3206 Group Project

(a) Description

The course is to allow students, in groups, to integrate the knowledge acquired over the previous five semesters into solving a non trivial problem through a computer application. Emphasis will be put on the systematic development methodology, the documentation of the development process, and how well the developed system address the problem to be solved. Non functional attributes like robustness, usability, security and reliability will also be tested.

(b) Aims

The aim of the course is to give students experience in

- Group and collaborative work
- Proper procedures in development of computer systems
- Proper documentation of the software development process
- Development of big not trivial computer projects.

(c) Teaching and Learning pattern

Students will be under the supervision of a member of staff (at least at a rank of Assistant lecturer). The supervisor will guide them in the day today progress of the project. When the supervisor feels the students have addressed the problem at hand, (s)he will sign off their report and recommend them for examination.

(d) Indicative content

The content is to be determined by the students under the guidance of the supervisor.

(e) Assessment Method

Students will submit the signed report for examination to the department. The department will appoint an examination panel of at least 5 people who will evaluate the report as well as testing the system. Students will be required to present their work and answer any questions from the panel. Each member of the panel will award a mark depending on his/her view on the worthiness of the developed application.

3.6.3 CSC 3207 Computer Security

(a) Description

Computer security is a branch of technology concerned with digital security or information security applied to computers. Since the largest part of the computer that users interact with is software, computer security pays big attention to development of secure software.

(b) Aims

The aims of the course are:

- To introduce students to threats faced by computers in the connected digital world.
- To introduce students to techniques that are used to protect computers against various threats.

(c) Teaching and Learning Patterns

The teaching pattern is by lectures, lab sessions and group projects

(d) Indicative content

- Digital security principles
- Hardware based security mechanisms
- Secure operating systems
- Security architecture
- Security by design
- Secure coding (Software Security)
- Access Control Lists
- Security Applications

(e) Assessment method

Assessment will constitute Practical assignments on at least 5 chapters of the course and written course work (20%) and written Exam (60%).

(f) Reading list

- (a) Ross J. Anderson: Security Engineering: A Guide to Building Dependable Distributed Systems, Willey 2001.
- (b) Robert C. Seacord: Secure Coding in C and C++. Addison Wesley, 2005.

3.6.4 BIT 3200 Business Intelligence and Data Warehousing

(a) Description

This course covers techniques and software tools that can assist management that deals with large amounts of data in management and business decision making. The course covers the fundamental differences between databases and data warehouses, the techniques of developing data warehouses as well as manipulating them to generate business strategic decisions.

(b) Aims

The aims of the course are

- To give students the understanding on the role and operation of data warehouses
- To equip students with skills of developing data warehouses
- To equip students with skills of maintaining existing data warehouses
- To equip students with skills of manipulating data warehouses to generate information for business decision making

(c) Learning outcomes

By the end of the course, the student should be able to

- Distinguish the roles of a database from that of a data warehouse
- Develop a data warehouse
- Populate and manipulate a data warehouse

(d) Teaching and Learning Pattern

Teaching will be in form of class lectures, tutorials, lab demonstrations as well as class presentations.

(e) Indicative Content

- Data warehouse concepts: partitioning, granularity, record of source, and meta data
- Building viable decision support environments.
- Architect development,
- Data migration and integration,
- Use of operational data stores, and transactional systems.

- (f) Assessment method Assessment will be in form of tests and practical assignments (40%) and final written examination (60%)
- (g) Reading lists
 - (i) Data warehousing fundamentals by Paulraj Ponniah Wiley 2001.

3.6.5 BSE 3202: Distributed Systems Development

1. Description

This course gives students theoretical and practical skills on development of distributed systems and applications. This include distributed-system-specific challenges like reliability and robustness.

2. Aims

The aim of the course are to equip students with skills of developing distributed systems

3. Teaching and Learning patterns

Teaching will be by class lectures and laboratory practicals/demonstrations

4. Indicative content

- Event-driven software architectures,
- Distributed object computing,
- Development, documentation and testing of distributed applications
- Techniques for reusable, extensible and efficient software systems
- Maintainability and concurrence in distributed systems
- Abstraction based on patterns and object-oriented techniques

5. Assessment Method

Assessment will be in form of tests and (practical) assignments (40%) and final examination (60%)

6. Reading list

- (i) S. Tanenbaum and M. V. Steen, Distributed Systems: Principles and Paradigms, Second Edition, Prentice Hall, 2006.
- (ii) R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, John Wiley & Sons, 2001.

4 Resources and Infrastructure

The Department of Computer Science and the Faculty of Computing and Information Technology have the enough resources and infrastructure to sufficiently run the revised program.

4.1 Staff

The Faculty of Computing and Information Technology has a big pool of staff who can competently teach the courses. The department also relies on staff from other departments like Information Technology, Information Systems and Networks.

4.2 Lecture Space

Initially, the Faculty of Computing and Information Technology housed in a 2,500 square meter building (Block A). In January 2009, a new 12,000 square meter building (Block B) was officially opened. The new building has lecture rooms together with general and specialized laboratories. The two buildings sufficiently cater for all the lecture and lab space requirements for all the teaching in the faculty.

4.3 Computer Laboratories

The old and new faculty buildings have general laboratories (strictly for students practice), teaching laboratories and specialized laboratories. These laboratories are shared among the departments of the faculty and are scheduled by the ICT services unit. Currently, the faculty has approximately 2000 computers and 5000 students. This leads to a student to computer ration of 1: 2.5 which is adequate for the practical components of the curriculum.

4.4 Software

On top of the physical computers, students need software for the different practical sessions. Different computers are installed with different software depending on their focus. Most of the software is available as free distributions for academic purposes. The faculty and department therefore have (and can access) enough software that can run the practical aspects of the program.

5 Quality Assurance

Several activities will be carried out as quality assurance measures so as to

- (a) Measure the general extent to which the required skills have been achieved
- (b) Ascertain the implementation of the methodological changes proposed
- (c) Create a feedback benchmark for possible future revisions in the curriculum

The following activities will be carried out in the process of monitoring and assuring quality in the proposed program.

5.1 Feedback from students enrolled

In the current set up, each class has 4 student representatives (2 day, 2 evening). These representatives are in constant contact with the Head of Department in case there are any quality related matters in a particular class. This set up is to be maintained.

At the end of the semester, samples of students are given questionnaires to respond to several quality related matters like staff punctuality, delivery mode, course content and the general perceived usefulness of the course unit. The Faculty of Computing and Information Technology is the process of creating a computerized system that will capture and analyze the data. With the computerized system:

- (i) Every student will be required to assess every lecturer teaching him/her, the sample space will therefore be increased
- (ii) No time will be required in the analysis of the results. Staff and faculty management will be able to get the feedback instantly
- (iii) Data will be easily archived and therefore the trend of staff performance in specific areas will be easy to visualize

5.2 Class meetings

The departmental management makes at least 1 meeting with every class every semester. In this meeting, general quality issues are addressed. Students are also given a chance to raise any questions that are answered and/or addressed by the department management. This set up will also continue

5.3 Use of ICT in availing lecture materials

Currently, Makerere University has the blackboard e-learning tool on its Intranet. Students in the Department of Computer Science have adequate access to computers. This creates a good environment for e-learning blended teaching. All courses in the new curriculum will be taught in a blended way. All course materials will be put on blackboard. Staff will, as much as possible, make use of e-learning facilities like discussion forum and drop boxes for assignments. This will increase student activity/participation and reduce staff effort (e.g. staff will not need to dictate notes). This in turn will, in turn, increase the material covered and taken in by the students.

5.4 Peer review

All members of staff will enroll (as students) to all classes taught in the department. They will therefore be able to view contents of courses taught by their peers. Staff will be free to advise fellow staff on the content, depth and presentation of materials. Consequently, for every course, students will access the best possible material in the view of all staff in the department not the course instructor

5.5 External examiners' reports

Like it is everywhere in Makerere University, student results are reviewed every semester by a senior external academician. This is to bring a 'foreign view' of the quality of the program. External examiners write reports on their view of the curriculum/examinations. Some recommendation can be implemented immediately while others have to be implemented in a longer term. The department will make the maximum possible use of external examiners' reports as a means of assuring quality in the revised program.