

<b>Course #:</b>	<b>CSCI-C 201</b>
<b>Course Title:</b>	<b>Computer Programming II</b>
<b>Course Type:</b>	Required core
<b>Prerequisites:</b>	C101 Computer Programming I
<b>Credits:</b>	4
<b>Text Book:</b>	Absolute C++, Second Edition, Walter Savitch, Addison Wesley
<b>References:</b>	Handouts
<b>Current Catalog Description:</b>	Fundamental forms and concepts of computer science, including top-down design, data structures, structured control flow, modular programming, recursion, and standard algorithms. Programming language concepts will be illustrated with C++.
<b>Course Goals</b>	<p>The student who completes this course:</p> <ol style="list-style-type: none"> <li>1. Will be proficient in advanced programming techniques such as those covered in advanced C++ .</li> <li>2. Will be proficient in constructing program that use elementary data structures, use of pointers, use of dynamically allocated memory such as linked list</li> <li>3. Will be proficient in the use of data files</li> <li>4. Will be proficient in the construction of classes and other object oriented facilities such as inheritance, overloading, overriding.</li> </ol>
<b>Major Topics Covered in the Course</b>	<ol style="list-style-type: none"> <li>1. Structs</li> <li>2. Bitwise operators {and the internal representation of data}</li> <li>3. More on pointers</li> <li>4. Multidimensional arrays</li> <li>5. Dynamically allocated arrays: new and delete operators</li> <li>6. Register storage class, How memory is allocated for a C/C++ program.</li> <li>7. Linked lists</li> <li>8. C style I/O, scanf and print</li> <li>9. More standard library functions, qsort, bsearch, clock, memmove, etc.</li> <li>10. Inline functions</li> <li>11. Function overloading</li> <li>12. Function templates</li> <li>13. More on parameter passing: const parameters, function parameters.</li> <li>14. Command line parameters.</li> <li>15. Typedef</li> <li>16. Conditional compilation;</li> <li>17. Separate compilation.</li> </ol>

	<p>18. Files</p> <p>19. Classes</p> <ul style="list-style-type: none"> <li>- member functions,</li> <li>- templates</li> <li>- implementing abstract data types</li> <li>- dynamic classes</li> <li>- constructors and destructors</li> <li>- inheritance</li> <li>- virtual functions</li> </ul>																		
<b>Laboratory projects (specify number of weeks on each)</b>	No closed laboratory																		
<b>Estimate Curriculum Category Content (Semester hours)</b>	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Area</th> <th>Core</th> <th>Advanced</th> </tr> </thead> <tbody> <tr> <td>Algorithms</td> <td>8</td> <td></td> </tr> <tr> <td>Software Design</td> <td>25</td> <td></td> </tr> <tr> <td>Comp. Arch.</td> <td>2</td> <td></td> </tr> <tr> <td>Data Structures</td> <td>5</td> <td></td> </tr> <tr> <td>Prog. Languages</td> <td>30</td> <td></td> </tr> </tbody> </table> <p>Additional hours may be dedicated to curriculum categories not listed above. For example explanation of concepts and theories. Discussion of social and ethical issues, discussion of inter personal relationships and working within groups.</p>	Area	Core	Advanced	Algorithms	8		Software Design	25		Comp. Arch.	2		Data Structures	5		Prog. Languages	30	
Area	Core	Advanced																	
Algorithms	8																		
Software Design	25																		
Comp. Arch.	2																		
Data Structures	5																		
Prog. Languages	30																		
<b>Oral and Written Communications</b>	Not a course objective.																		
<b>Social and Ethical Issues</b>	Issues of software license, the difference between commercial, public, and open source software, the GNU project and associated license, 30 minutes.																		
<b>Theoretical Content</b>	Not a course objective.																		
<b>Problem Analysis</b>	The student is taught how to analyze a problem and the correct steps to solving a problem and writing an algorithm.																		
<b>Solution Design</b>	<p>The design steps would be the following:</p> <ol style="list-style-type: none"> <li>1) Understand what the question or problem is asking.</li> <li>2) Identify what is needed by the problem; all inputs, outputs and equations.</li> <li>3) Determine the constraints or limits on the problem.</li> <li>4) Write an algorithm to solve problem.</li> <li>5) Translate algorithm into a program.</li> <li>6) Run and test program</li> </ol>																		
<b>Prepared By</b>	Scheessele, Hakimzadeh, Holloway,																		