**Montana Tech C Programming Algorithmic Language Framework V1.2**Version 1.2  
March 29, 2015

A. Frank Ackerman

*Computer Science  
Montana Tech*

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comment |
| 1.0 | 01/25/15 | Frank Ackerman | Initial version |
| 1.1 | 02/15/15 | Frank Ackerman | Added **Call** |
| 1.2 | 03/29/15 | Frank Ackerman | Added **Link** |
|  |  |  |  |
|  |  |  |  |

**Montana Tech *C Programming Language I* students:**

This standard encapsulates Dr. Ackerman’s decades of experience in the software industry, the IEEE software engineering standards, and many suggestions from various texts. You are a invited to participate in the continuing evolution of this by studying it critically and making suggestions for its improvement and correction.

Purpose

The purpose of this document is to define a base language for the expression of algorithms that are to be translated into C programs or functions. This base language is intended to be sufficient for expressing algorithms that can be translated into C language computer programs or functions, but yet be free of the many details involved in using C directly to develop an algorithm.

Introduction

This document is intended to provide *C Programming Language* students with an easy-to-use and easy-to-understand method for unambiguously describing algorithms that can be translated into C programs, and that can be unambiguously mentally “executed” by fellow students

The constructs defined in this standard are divided into the following categories:

### **3. Sequential Constructs**

**4. Selection Constructs**

**5. Repetition Constructs**

Overview

A C language algorithm that can be easily translated into a C program consists of a sequence of algorithm language constructs. Each construct begins with one of the “starter words” (initial letter capitalized) from the list below. Each element may begin with a label of the form A*dd* that is attached to the construct. When this is done all algorithm statements should be tabbed over so that the Add labels are all aligned at the left margin.

As much as possible, language elements should be label in the sequence A00, A01, A02, .... . As the algorithm develops and new elements need to be inserted, these labels may be extended by adding a suffix of a, b, c, ...

The list of algorithm constructs given in the next section is the heart of this document. To facilitate clear intent, and to make an algorithm abstractly executable, only the listed constructs may be used.[[1]](#footnote-1)

The phrases after the keywords are not explicitly defined but should clearly describe the intention or meaning of the construct for that algorithm, as should any optional comment following a terminating semicolon or closing brace. Since algorithm text may be included in a source file, which almost always uses a single fixed width font, ordinary English words should not be used to name objects. Use compound names that clearly reference the object or attribute. For example, use itemCount instead of count to reference an item count.

# Sequential Constructs

|  |  |
| --- | --- |
| **Accumulate** *text*; | to describe a += accumulation operation |
| **Add *text;*** | to add a numeric quantity |
| **Calculate** *text*;  **Compute** *text*; | to describe a calculation that is more complex than incrementing or decrementing a object or attribute. |
| **Call** *functionName* | Invoke a function |
| **Convert** *from one form to another***;** | usually to describe changing a type or converting a value, e.g., miles to kilometers |
| **Decrement** varName *by amount*; | to describe that a value is being decremented (the amount is 1 if not stated)  This construction should always explicitly reference an object or attribute |
| **Display** *text***;** | Usually implemented at  printf(*format-string*, *scalarDataItem*, ...) |
| **Increment** varName *by amount*; | to describe that a value is being incremented (the amount is 1 if not stated.  This construction should always explicitly reference an object or attribute |
| **Link** *text;* | in a linked list, adjust pointers appropropriately. |
| **Obtain** *text;* | Prompting a user for input  We will always terminate a prompt with ‘> ‘ |
| **Read** *text***;** | Read text data from user or file |
| **Reset** *varName* to *text*;  or  **Reset** *funcName for* *text*; | to change the value of an object or attribute back to something it was before  or  call a function to reset static variables |
| **Return** *[object value];*  or  **Return** *to invoker* | to return from a function;  or  to exit a program, return 0 for a normal exit and 1 for an error exit. |
| **Set** varName to *text*; | to change the value of an object or attribute |

# Selection Constructs

|  |  |
| --- | --- |
| **If** (*condition*) {  *body of if*  **}//If** *optional comment* | for code that is to be executed only if *condition* is true |
| **If** (*condition*) {  *body of if*  **}//If**  **Else {**  *body of else*  **}//Else** *optional comment* | Two-way branch |
| **If** (*condition*){  *body of if*  }//if  **Else If *(condition)*{**  *body of else if*  **}//Else If**  **Else {**  *body of else*  **}//Else** *optional comment* | Multi-way branch |

# Repetition Constructs

|  |  |
| --- | --- |
| **Break;** | for code break |
| **For** (*iteration expression*){  *body of for*  **}//For** *optional comment* | to describe loops that iterate across a sequence of items |
| **While** (*condition*) {  *body of loop* **}//While** *optional comment* | to describe a loop that will not be executed even once if *condition* is not meet  For an loop in which the exit condition is given by a break in the body of the loop, *condition*, is forever. |

# Example

Problem statement: For any Fahrenheit temperature value in [-1010, 1010] compute the equivalent Centigrade temperature value.

An algorithm that gives this result is:

A01 **Display** starting salutation;

A02 **Obtain** inFahr\_deg from user;

A03 **Compute** outCent\_deg from inFahr\_deg;  
A04 **Display** outCent\_deg;

A05 **Display** ending salutation;

1. When we run into situations for which we have not yet defined a good algorithm “starter word” we will extend this standard to cover this situation. [↑](#footnote-ref-1)