# Home Construction Modeling Language (HCML)

Joe Janik – UIN: jj2543 – Email: jj2543@columbia.edu
W4115 Programming Languages and Translators
Proposal Assignment - Fall 2010 Semester

## Objective:

To create a language in which a user can model home construction design plans and build a catalog of supplies needed to complete work. Design plans for room configurations will be programmed and compiled where the build parameters are checked for house type and allocated square footage. Then a listing of supplies will be generated for purchasing along with a 2D layout of the floor plan.

## Motivation:

Being a first-time home buyer and then rehabbing it, I've ran into occasions where proper planning of the space layout would have saved me time, money, and frustration. Modelin g portions of the construction project (such as wall framing and drywall) using HCML wi ll aid in visualizing the layout of the working space and generating the correct amount of supplies needed to complete the job (number of 2x4s and sheets of drywall for a 10x20ft wall section).

## Features:

All room objects will be modeled: entry ways, windows, ceilings, flooring, closets, etc... Command set will be small enough to model room dimensions, perform calculations for b uild supplies based on project definitions, and output patterns for modeling the room obje cts. Concepts and commands will be utilized in a human friendly manner allowing your “average Joe” to be a Pro HCML Carpenter.

## Language Overview:

Comments will be implemented in a similar fashion to the C programming language. Sin gle-line comments will begin with “//” and block comments will be enclosed with “/*” at the start of the comment and “*/” at the end of the comment.

Language flow control and layout will be implemented using the standard grammatical symbols:

> “ ; ” Semi-colon to mark the end of a statement
> “ , ” Comma to separate arguments within a function
> “ ( ) ” Left and Right parentheses for grouping of arguments
> “ [ ] ” Left and Right Square brackets for arrays
> “ { }” Left and Right Curly braces for grouping block statements with their bodies

Iteration, loops, and evaluation statements will be modeled off the C programming language implementations and consist of the following:

- “if then else” for evaluation rules for use with comparators
- “for” performing a body of code a set number of times based on the evaluation statement
- “while” similar to “for” but with a differing evaluation statement format

Blank spaces and tabs will be ignored until the end of line character " \n " is detected.

## Reserved keywords and functions:

- "build" - create project for construction design
- "construct" - create object such as wall, ceiling, floor, door, and window
- "demo" - destroy and delete object
- "position" - east, west, south, and north
- "wall" - has dimension parameters height, length, and position
- "ceiling" - has dimension parameters length and width
- "floor" - has dimension parameters length and width
- "door" - has dimension parameters length, height, and position
- "window" - has dimension parameters length, height, and position
- "generate" - creates list of raw building supplies needed to complete project

## Reserved data types:

- Text - Strings of characters
- Int - Integer numbers
- Float - Floating point numbers
- Boolean - True or False

## Operators:

- " + " - for addition
- " - " - for subtraction
- " / " - for division
- " * " - for multiplication

The operator precedence and associativity will be the same as in the C programming language.

Comparators such as “ == ” double equal sign for comparing equality, “ > ” greater-than-sign and “ < ” for evaluating data ranges will be used in HCML.

- " > " - for greater-than comparisons
- " < " - for less-than comparisons
- " == " - for an equation comparison
- " >= " - for greater-than-or-equal-to comparisons
- " <=" - for less-than-or-equal-to comparisons

## Example code:

** Note that format is preliminary and subject to change - the language is syntax and declarations are defined here as a proposed solution however can change dependent on feedback from Professor and TAs. **

```
// stand up project
build (masterbedroom)

// build wall of dimension 10ft height x 20ft length on south and north sides of room
build (masterbedroom.wall, south, north){
        int h = 10;
        int l = 20;
}

// build wall of dimension 10ft height x 12ft length on east and west sides of room
build (masterbedroom.wall, east, west){
        int h = 10;
        int l = 12;
}

// build floor and ceiling same dimensions 20x12
build (masterbedroom.ceiling, masterbedroom.floor){
        int l = 12;
        int w = 20;
}
```

```
// destroy south wall built in previous command
demo(masterbedroom.wall, south);

// generates raw building materials needed for project
generate (masterbedroom);
```

Output is a text file with raw building materials:

- Qty x (2x4x10s studs)
- Qty x drywall sheets (8x4 panels)
- Qty x plywood (8x4 panels)