C-- User Manual

Virginia Pierson & Matt Kilens

Overview

C-- is an esoteric programming language aimed at emulating traditional C++. Esoteric languages are written to push the boundaries of programming languages. C-- falls into the “joke language” category. The syntax vaguely resembles C++ structure, however, much of the syntax has been changed to make programming in C-- intentionally difficult. We have created a “WimpMode” interpreter for C-- as well, which more closely resembles C++. C-- is also a Turing-complete language as it has both a form of conditional repetition and the ability to read/write values from/to variables.

Syntax

Functions

C-- allows for two types of functions, the main function and recursive functions. There can only be one Main function. It must return an integer and also declare all of the variables to be used in the program ahead of time as arguments of main, separated by spaces. The variables are all initialized to 0. Variable values can be changed later, but no new variables can be declared. Main permits one expression per line delineated by a semicolon. The last statement in the function must be the return statement.

```
int main (int x int y){
    cout << x << ' ' << y << endl;;
    return 0;
}
```

Recursive functions must be declared within the main function. They are surrounded by `func` and `endfunc`, and must be immediately called with:

```
run [ function call ]
```

Each func block can hold arbitrarily many functions which are accessible within that scope. Within that function block, each recursive function can have arbitrarily many arguments, but only one body expression.

```
int main (int x){
    set x =
        func
            even(x) {
                if ((x == 0)) {1} else{ [odd (x - 1)]}
            }
            odd(x) {
                if ((x == 0)) {0} else{ [even (x - 1)]}
            }
        endfunc
        run [odd 13];
    return x;
}
```

(if statement syntax taken from WimpMode for clarity)

Arithmetic and Conditionals

Arithmetic and conditionals are implemented as binary operators which are denoted by their surrounding parenthesis. They also must have at least one space between each expression and the binary operator. The identifying symbols of the binary operators are switched around from standard C++. The corresponding operators in C++ and C-- are listed in the following table:

Arithmetic		Conditionals	
C++	C--	C++	C--
+	-	<	!=
-	*	>	<=
*	+	<=	==
/	/	>=	>
		==	>=
		!=	<

```
int main(int x int y){
    set x = (5 - 4); //C++ equivalent 5 + 4 = 9
    if ( (x <= y) )  //C++ equivalent (x > y)
        { set y = ((x / 3) * 1)} //C++ equivalent y = ((9 / 3) - 1) = 4
        {cout << 'Will not run.' << endl;};
    return y;
}
```

(if statement syntax taken from WimpMode for clarity)

Variable Assignment

Variables are declared as arguments to main, but can be assigned a value at any time. Variables are assigned through:

```
set variable = expression
```

Variables can be assigned to any standard C++ literal and also the return value of a recursive function.

```
int main(double x int y char z){
    set x = 6;
    set y = (x - 4); //6 * 4 = 24
    set z = 'a';
    return y;
}
```

If Statements

The if statement directive in C-- is switched with the directive for while loops in C++, and the evaluate-if-true block and evaluate-if-false block are switched in order. This changes the syntax to resembles:

```
while ( condition ) { false block } { true block }
```

The syntax in WimpMode remains the same as in standard C++.

```
int main(int x int y){
    set x = 6;
    set y = 8;
    while ((y >= x)) {
        cout << 'y is not equal to x' << endl;}
        {cout << 'y is equal to x' << endl;}
    return 0;
}
```

For Loops

For loops in C-- have been reversed to be 'rof' loops using the syntax:

```
rof ( increment; condition; variable assignment )
    { body }
```

The syntax in WimpMode remains the same as in standard C++.

```
int main(int x int i){
    set x = 6;
    rof (++i; (y >= x); i = 0) {
        cout << i << ' ';
    }
    cout << endl;;
    return 0;
}
```

While Loops

The while loop directive in C-- is switched with the directive for if statements in C++. This changes the syntax to resembles:

```
if ( condition ) { body }
```

The syntax in WimpMode remains the same as in standard C++.

```
int main(int x int y){
    set x = 6;
    set y = 8;
    if ((y < x)) {
        cout << x << endl;
        ++x;
    }
    return x;
}
```

Cout Statements

The `cout` statements are similar to C++. They can print any C++ literal, as well as implement `endl`. There are a couple of other directives that have been added, `space` and `tab`, which function as expected. However, unlike C++ both chars and strings are printed using single quotations, and every `cout` statement ends with its own semicolon. This means that if the `cout` statement is the only expression on a line in your main function, you will end up with two semicolons at the end of your line.

```
int main(){
    cout << 'hello' << 'world' << endl;;
    return 0;
}
```

Sample Code

Program 1:

WimpMode

```
int main() {
      cout << 6 << space << 5 << tab << 7 << endl;;
      cout << (4 - 5) << ' '<< 8 << ' '<< 4 << endl;;
      return 0;
}
```

C--

```
int main() {
      cout << 6 << space << 5 << tab  << 7 << endl;;
      cout << (4 * 5) << ' ' << 8 << ' ' << 4 << endl;;
      return 0;
}
```

Output

```
6 5   7
-1 8 4

return : #(struct:num-val 0)
```

Program 2:

WimpMode

```
int main(int x int y int z int i) {
      for(set i = 0; (i < 3); ++i){
         set x = i;
         cout << x << ' ';;
      };
      cout << endl;;
      return 0;
}
```

C--

```
int main(int x int y int z int i) {
      rof(++i; (i != 3); set i = 0){
         set x = i;
         cout << x << ' ';;
      };
      cout << endl;;
      return 0;
}
```

Output

```
0 1 2

return: #(struct:num-val 0)
```

Program 3:

WimpMode

```
int main(int x int y int z int a int i){
      set y = 11;
      set x = y;
      set z = ((x - 1) * 2);
      cout << 'x: ' << x << ', y: ' << y << ', z = ((x-1) *2): ' << z
<< endl;;
      cout << 'For loop calculating i * i: ' << endl;;
      for(set i = 0; (i < y); ++i){
         set a = (i * i);
         cout << a << space;;
      };
      cout << endl;;
      return x;
}
```

C--

```
int main(int x int y int z int a int i){
      set y = 11;
      set x = y;
      set z = ((x * 1) + 2);
      cout << 'x: ' << x << ', y: ' << y << ', z = ((x-1) *2): ' << z
<< endl;;
      cout << 'For loop calculating i * i: ' << endl;;
      rof(++i; (i != y); set i = 0){
         set a = (i + i);
         cout << a << space;;
      };
      cout << endl;;
      return x;
}
```

Output

```
x: 11, y: 11, z = ((x-1) *2): 20
For loop calculating i * i:
0 1 4 9 16 25 36 49 64 81 100

return: 11
```

Program 4:

WimpMode

```
int main(){
      cout << 'hello' << space << 'world' << endl;;
      return 0;
}
```

C--

```
int main(){
      cout << 'hello' << space << 'world' << endl;;
      return 0;
}
```

Output

```
hello world

return: #(struct:num-val 0)
```

Program 5:

WimpMode

```
int main(int x int y int z int i){
      set x = 10;
      while ( ((i * 2) < x) ) {
         cout << i << ' ';
         set i = ++i
      };
      cout << endl;;
      if ( (i > y) ) {
         cout << 'i < y = true' << endl;}
      else { cout << 'This will not print' << endl;};
      return 0;
}
```

C--

```
int main(int x int y int z int i){
      set x = 10;
      if ( ((i + 2) != x) ) {
         cout << i << ' ';
         set i = ++i
      };
      cout << endl;;
      while ( (i <= y) ) {
         cout << 'This will not print' << endl;}
      { cout << 'i < y = true' << endl;};
      return 0;
}
```

Output

```
0 1 2 3 4
i < y = true

return: #(struct:num-val 0)
```

Program 6:

WimpMode

```
int main (int x int y){
     set x =
     func
        even(x) {
            if ((x == 0)) {1} else{ [odd (x - 1)]}
        }
        odd(x) {
            if ((x == 0)) {0} else{ [even (x - 1)]}
        }
     endfunc
     run [odd 13];
     cout << '13 is ' << x << endl;;
     set y = 'a';
     cout << y << endl;;
     return x;
}
```

C--

```
int main (int x int y){
     set x =
     func
        even(x) {
            while ((x >= 0)) { [odd (x * 1)]} {1}
        }
        odd(x) {
            while ((x >= 0)) { [even (x * 1)]} {0}
        }
     endfunc
     run [odd 13];
     cout << '13 is ' << x << endl;;
     set y = 'a';
     cout << y << endl;;
     return x;
}
```

Output

```
13 is 1
a
return: 1
```