Burt Rosenberg

C Syntax

OUT: 15 SEPTEMBER, 1995

- CONSTANTS.
  - *Integer* such as `1, 0, 14, 0x0A`.
  - *Characters* such as `'A', 'B', '\0'`.
  - *Strings* such as `"Hello World!\n", "A"`.
- VARIABLES, such as `u, i, u_and_i_are_variables`.
- EXPRESSIONS. Constants and variables are expressions. So are expressions connected together by operators and using parentheses to group operators. The operators are:
  - *Arithmetic:* `+, -, *, /, %` and unary `-` meaning "negative."
  - *Assignment:* `=, +=, -=, *=, &=, |=, ++, --`
  - *Access:* access an array element, `[ ]`, access a structure element, `.` or `->`, call a function, `( )`, access the target of a pointer, `*`.
  - *Logical:* `==, !=, <, <=, >, >=, &&, ||, !`
  - *Bitwise:* `&, |, <<, >>, ~, ^`
  - *Utility:* the cast `( )`, the address operator `&` and the `sizeof` operator.

  Examples: `x+1, (x+1), (1+3), x=((y+z)>2)+1, (*x[100]).field, f(1)`
- STATEMENTS.
  - Expressions followed by a semicolon are statements, such as `x=y+1;`
  - *Compound Statements:* a series of declarations, perhaps none, followed by series of statements, perhaps none, all enclosed in a pair of curly braces.
    Examples: `{ x=1; y=2; }, {int i; i=1; }, {}, {;}`
  - The *If Statement:* `if ( Expr ) Statement`
    Examples: `if (x>y) x=y+1; if ((x+1)==y) { x=1; y=2; }`
  - The *If-Else Statement:* `if ( Expr ) Statement else Statement`
    Examples:

```
if (x>y) x=y+1; else { x=1; y=2; }
if ((x+1)==y) { x=1; y=2; } else x=y+1;
if (x=(y>1)) x++ ; else y-- ;
if (x==y) { x=2; y=3; } else { x=3; y=2 }
if (x==y) ; else { do_only_me++ }
```

  - The *While Statement:* `while ( Expr ) Statement`
    Examples: `while (a>0) a-- ; while ( ++a<100 ) ;`

- The *For Statement:* `for ( Expr1 ; Expr2 ; Expr3 ) Statement`
  Note: This statement is equivalent to `Expr1 ; while (Expr2) { Statement Expr3; }`
  Examples: `for(i=0;i<10;i++) a[i]=0 ;`
- The *Switch Statement:* `switch (Expr) Compound-Statement`, where the Compound Statement generally has case-labeled lines: `case Expr: Statement`, and the special `default: Statment`.
  Example:

```
switch( string[0] )
{
    case 'y': printf("Yes\n") ;
              break ;
    case 'n': printf("No\n") ;
              break ;
    default:  printf("string[0] is neither y nor n\n") ;
}
```

- Other statements include the *Break:* `break;`, the *Continue:* `continue;`, the *Return* of two forms: return with value: `return(Expr);`, return without value: `return;`, the *Conditional Evaluation:* `( Expr1 ) ? Expr2 : Expr3 ;`, and the *Null:* `;`.

- DECLARATIONS. These are allowed as the first elements of a compound statement or at the outermost level of the file (outside of all curly braces). They are of the form: *storage-class type name-list* `;`.
  - Storage-class: `extern, static, typedef` are the most frequently used.
  - Types: `char, int and double` are the most frequently used, as well as a structure specifier or a typedef name. Less frequently used are float, unsigned char, unsigned long, and a union specifier.
  - Name-list: a comma separated list of possibly initialized declarators of the following kinds: *identifier, pointer, function* or *array.* Parenthesis are used to nest declarators, forming very complex types.
    - *identifier*: a name.
    - *pointer*: `*` followed by a declarator or name.
    - *function*: a declarator or name followed by `()`
    - *array*: a declarator or name followed by `[optional-integer]`, where the optional integer constant indicates the size of the array.
  - Structure specifier: we come back to talk about these here because they are complicated. These are of new main forms
    - To declare new structure type: `struct optional-structure-tag { declarations }` The declarations are a sequence of declarations exactly as described by the section, except that the storage-class tags do not apply.
    - To recall a defined structure type: `stuct structure-tag`.

– Typedef: The storage class `typedef` declares a name for a new type. Write the declaration as if you were defining a variable, however the storage class `typedef` will assign to the variable the resulting type, which can then be used in further declarations.

Examples:

```
int i ; int i, j; int *pntr_i ; int i, *pntr_i ;
double x[100] ; double y, x[100], *pntr ;
char *array_of_10_pntrs_to_char[10] ;
char (*Pntr_to_an_array_of_10_char)[10] ;
struct { int i, j } a_struct, another_struct ;
struct S { int i ; double x ; } Array_of_S[10], *Pntr_to_S ;
struct S yet_another_S_struct, ten_more_such[10] ;
typedef LITTLE_ARRAY[10]; LITTLE_ARRAY i_am_an_array ;
```

- FUNCTION DEFINITION. These are allowed only at the outermost level of the program file. A function named `main` is special: there must be one and only one such function for each program. The syntax is *function-declaration(prototype) function-body.*
  – *Function Declaration with prototype:* Use the declaration syntax to give the name and return type of the function, and within the parentheses, give a comma separated list of declarations giving the name and types of the arguments. The storage class is ignored for arguments, and only the classes extern and static are generally used for the function itself. In the case of functions "static" means a function private to the current file and "extern" means a function whose name is exported to other files in a project.
  – *Function Body:* A compound statement.

Examples:

```
 int f(int i) { return(i) ; }
 int g( int a[], int n )
 {
   int j, sum ;
   sum = 0 ;
   for ( j=0;j<n;j++ )  sum+= a[j] ;
   return(sum) ;
 }
char * h(char * s)
{
   while (*s!=\'0')
   {
     if (*s==',') break ;
     s++ ;
   }
   return(s) ;  /* a pointer to a character, not *s! */
}
```

- PREPROCESSOR.
  - *#include:* Read in text from the named file. If enclosed in double quotes, the file name is used exactly. If enclosed in pointy brackets, the include file is looked for in the location of system wide include files. The preprocessor is not the compiler — an ending semicolon is not used.
    Examples:
    ```
    #include<stdio.h>
    #include "myheaderfile.h"
    ```
  - *#define:* Makes the first word that follows an alias for the rest of the line. Macros are also created using define.
    Examples:
    ```
    #define PI 3.14159
    #define TRUE 0==0
    ```
  - *Conditional Compilation:* These controls are used for making C code that compiles on many different platforms, using code pieces that adapt to the system's configuration. I mention them here because they are useful for "commenting out" large blocks of code when debugging. In C, comments of the `/* */` type cannot be nested. Instead, use:

```
#if 0
     The compiler will skip over
     all of this because what
     follows the #if is value 0.
     The compiler begins again
     after the #endif ...
#endif
```