LENGUAJE DE PROGRAMACIÓN SIMPLEX

Simplex es un lenguaje de programación orientado a la enseñanza, como su nombre lo indica se busca que sea simple y sencillo de utilizar, permitiendo que el programador comprenda los conceptos básicos de programación, como la estructura de un programa básico, tipos de datos, uso de variables, expresiones aritméticas, etc. antes de profundizar en conceptos de bajo nivel, como el manejo de punteros, reservación de memoria, etc.

En lo que se refiere a sintaxis, está fuertemente basado en el lenguaje C, los bloques de instrucciones son delimitados por llaves y la declaración y el tipo de variables son muy parecidas también. La diferencia más marcada es la eliminación del punto y coma para indicar el fin de una instrucción, ya que esto muchas veces puede llevar a cometer errores muy difíciles de detectar.

Es un lenguaje case sensitive, de alto nivel y utiliza el paradigma de programación imperativo.

También se requiere de punto de entrada del programa el cuál es la palabra reservada **main** seguida de un bloque de instrucciones delimitado siempre por llaves que constituyen el programa.

TIPOS DE DATOS SOPORTADOS

Los tipos de datos soportados actualmente por el lenguaje son los siguientes:

TIPO	LONGITUD MÁXIMA
Int	16 bits (65,536 valores)
Float	16 bits (precisión de 4 decimales)
Bool	1 bit
Char	1 Byte
String	200 Bytes
Array	100 posiciones

SimpleX

PALABRAS RESERVADAS

En total son 40 palabras reservadas

and	**compare**	**float**	**or**	**sort**
array	**concat**	**for**	**pow**	**string**
average	**else**	**function**	**print**	**substring**
bool	**even**	**if**	**procedure**	**switch**
break	**exist**	**int**	**read**	**true**
case	**factorial**	**main**	**ref**	**until**
char	**false**	**null**	**repeat**	**while**
closeFile	**file**	**openFile**	**return**	**sort**

CARACTERES ESPECIALES

OPERADORES:

Aritméticos: +, -, *, /, %

Lógicos: and, or, >, <, >=, <=, !, !=, ==

Asignación: =

Agrupación aritmética: (,)

Agrupación de instrucciones: {, }

DECLARACIÓN DE VARIABLES

int a = 0, b

bool condicion = false

char car = ‘c’

file archivo1

float b = 123.8821

array arr<int> [10]

string cad = “Hola mundo”

CONSTANTES

Lógicas: true, false

INSTRUCCIONES DE SELECCIÓN

IF

Uso:

```
if (condicion) {
        <instrucciones>
}else{
        <instrucciones>
}
```

SWITCH

Uso:

```
Switch (condicion){
        case: <valor>
                <instrucciones>
                break
        case: <valor>
                <instrucciones>
                break
        .
        .
        .
}
```

EXIST

Uso:

```
exist (palabra, caracter)
```

INSTRUCCIONES DE ITERACIÓN

WHILE

Uso:

```
while (condición) {

}
```

FOR

Uso:

```
for (i=0 ; condición; inc){

}
```

REPEAT

Uso:

```
repeat {

 } until(condición de paro)
```

EXPRESIONES REGULARES

IDENTIFICADORES DE VARIABLES:

Letra (Letra + Número)*

Dónde:

Letra = [a-z|A-Z]

Número = [0-9]

Longitud máxima 32 caracteres.

NÚMEROS ENTEROS:

Número +

Dónde:

Número = [0-9]

NÚMEROS FLOTANTES:

Número + . Número +

Dónde:

Número = [0-9]

COMENTARIOS:

Una línea: #

 # Esto es un comentario

Bloque: /* *\

/* Esto es un
comentario de bloque *

MANEJO DE ARREGLOS:

Declaración: array arreglo1<int> [5]

array arreglo2<string> [10]

Uso: arreglo1[0] = 25

arreglo2[9] = “Ola ke ase”

DECLARACIÓN DE FUNCIONES:

function <nombre_funcion>(<parametros>) : <tipo de retorno>

Ejemplo:

Function sumar(int n1, int n2) : int

Dónde:

nombre_funcion = identificador

parametros = (<tipo> <nombre_parametro>) * Cada parámetro se separa por coma.

DECLARACIÓN DE PROCEDIMIENTOS:

Procedure <nombre funcion>(<parametros>)

Ejemplo:

Procedure incrementar(int cantidad)

Dónde:

nombre funcion = identificador

parametros = (<tipo> <nombre_parametro>) * Cada parámetro se separa por coma.

MANEJO DE PARÁMETROS:

POR REFERENCIA:

Declaración: Procedure incrementar (ref int cantidad)

Uso: incrementar(ref cantidad)

POR VALOR:

Declaración: Procedure incrementar (int cantidad)

Uso: incrementar(cantidad)

FUNCIONES PREDEFINIDAS:

- **print**(variable, buffer) Imprimir en un buffer (puede ser archivo o consola), sino se especifica buffer imprimirá en consola.
- **read**(variable, buffer)Entrada de buffer (puede ser archivo o consola) devuelve el valor leído, sino se especifica buffer leerá de consola.
- **sort**(). ordenar arreglos

- **concat**(string1, string2). concatenar string

- **compare**(string1, string2) string

- **factorial**(entero).

- **average**(arreglo de enteros)

- **pow**(base, exponente) n de un número

-**even**(entero) Determina si el número es par.

-**substring**(cadena, inicio, fin) Devuelve una subcadena del argumento

MANEJO DE ARCHIVOS

Para el manejo de archivos se utiliza un tipo especial, llamado **file,** en el cual se almacena un buffer devuelto por la función **openFile**, y las subrutinas predeterminadas print y read, con las cuales se puede leer o escribir en el archivo. Para cerrar este buffer se utiliza la función **closeFile**

Funciones para apertura y cierre de archivos:

-**openFile**(string ruta). Abrir archivo de texto.

-**closeFile**(File archivo). Cerrar archivo de texto.

EJEMPLOS

MANEJO DE ARCHIVOS

```
main {
        file archivo = null
        archivo = openFile("C:\archivito.txt")
        if (archivo == null){
                print("Hubo un error al abrir archivo")
        }else{
                print("Este es un ejemplo", archivo)
                closeFile(archivo)
        }
}
```

FUNCIONES Y VARIABLES

```
int c     #Variable global
function sumar(int n1, int n2) : int
main {
        int a =3
        int b = 5
        c = sumar(a, b)
        if (c<0){
                print("El número es negativo")
        }else{
                print("El número es positivo")
        }
```

```
}

function sumar(int n1, int n2) : int {
        return n1+n2
}
```