Extensiones para el lenguaje de programación C++
__ ec++ __

Propuesta realizada por Octulio Biletán.
Las nuevas palabras reservadas son: **entry**, **exit**, **access**, **forward**, **rewind**, **specific**, **random**, **loop**, **until** y **repeat**.
Las nuevas funciones agregadas son: **on()**, **off()**.
Un nuevo fichero de encabezamiento se agrega para contener las declaraciones de on() y off(): **<pointers.h>**

El programa traductor que se necesitaría para traducir de un programa fuente **.e.cpp** a otro programa fuente **.cpp** se llama **ec++**.
Sería invocado así:
	ec++ ejemplo1.e.cpp

Y obtendría el programa fuente ejemplo1.cpp.
Para construir el programa ejecutable se invocaría por ejemplo a Embarcadero C++ Builder:
	bcc32 ejemplo1.cpp

Y se obtendría el programa final ejemplo1.exe para ser corrido en un sistema Windows.
Para el caso de un sistema operativo con Linux se utilizaría la herramienta "GNU compiler collection".

[*]Ciclos

Ciclo do...loop
Las palabras reservadas **do** y **loop** pueden ser usadas de dos formas diferentes para constituir un ciclo repetitivo.
(1)	Usar:
	do
	{
		cuerpo;
	}loop;

(2)	Usar:
	loop
	{
		cuerpo;
	}

_____
Ciclo do...until
Las palabras reservadas **do** y **until** pueden ser usadas de dos formas diferentes para constituir un ciclo repetitivo.
(1)	Usar:
	do
	{
		cuerpo;
	} until(cond);

(2)	Usar:
	until(cond)
	{
		cuerpo;
	}

_____
Ciclo repeat(n)
	Usar:
	repeat(n)
	{
		cuerpo;
	}

_____

Ciclo access
Básicamente recorre secuencialmente una formación unidimensional de datos numéricos o alfanuméricos ya sea desde su primera posición o desde su última posición hasta la última posición o hasta la primera posición.
Otros recorridos: especificado por la posición y aleatorio.
Los tipos aceptados para los datos numéricos son: ’int’, ’int8’, ’int16’, ’int32’ e ’int64’.
Los tipos aceptados para los datos alfanuméricos son: ’char’ y ’wchar_t’.

Sintaxis:
	1.- access(< _vector_ >; forward | rewind | specific | random) { cuerpo; }
	2.- access(< _vector_[posición] >; forward | rewind) { cuerpo; }
	3.- access(< _vector_[posición_inicial : posición_final] >; forward | rewind) { cuerpo; }
	4.- access(< _vector_[p0], _vector_[p1], ... >; specific) { cuerpo; }
	5.- access(< _vector_ >; random) { cuerpo; }

Para el caso ’2’, el campo ’posición’ es un valor numérico entero y natural tanto positivo como negativo.
Los tipos aceptados para los números son: ’int’, ’int8’, ’int16’, ’int32’ e ’int64’.
Pueden usarse junto con los especificadores de tipo: short, long, signed y unsigned.
Para el caso ’3’, se permite hacer un recorrido hacia adelante o hacia atrás sobre un rango de posiciones.
Para el caso ’4’, se permite hacer un recorrido específico sobre posiciones concretas del vector.
Para el caso ’5’, se permite hacer un recorrido aleatorio sobre las posiciones del vector.

Ciclo access y forward
Recorre secuencialmente la formación unidimensional de datos numéricos o alfanuméricos desde la posición 0 hasta su

última posición.
Por ejemplo:

```
        int vector[] = { 4, 33, 23, 1, 0, 7, 99, -6, -62, 19 };

        access(vector; forward)
        {
            // Muestra el contenido del vector, posición vigente
            cout << *vector << endl;

            // Si en la posición vigente del vector contiene un 7
            // entonces se sale del ciclo 'access'
            if(*vector == 7)
                break;
        }
```

Ciclo access y rewind
Recorre secuencialmente la formación unidimensional de datos numéricos o alfanuméricos desde la última posición hasta la posición 0.
Por ejemplo:

```
        int vector[] = { 4, 33, 23, 1, 0, 7, 99, -6, -62, 19 };

        access(vector; rewind)
        {
            // Muestra el contenido del vector, posición vigente
            cout << *vector << endl;

            // Si en la posición vigente del vector contiene un 7
            // entonces se sale del ciclo 'access'
            if(*vector == 7)
                break;
        }
```

Para todas las instrucciones de ciclos se puede utilizar la palabra reservada 'break' para salir del ciclo.
Por ejemplo:

```
        int vector[] = { 4, 33, 23, 1, 0, 7, 99, -6, -62, 19 };
        int a = (sizeof(vector) / sizeof(int)) - 1;
        until(a == -1)
        {
          cout << vector[a] << endl;
          if(vector[a] == 7)
            break;
          a--;
        }
```

También es posible utilizar la palabra reservada 'continue' para continuar ciclando.
Por ejemplo:

```
        int vector[] = { 4, 0, 23, 7, 0, 0, 0, -6, -62, 19 };

        access(vector; rewind)
        {
            // Si en la posición vigente del vector contiene un 0
            // entonces continúa ciclando.
            if(*vector == 0)
              continue;

            // Muestra el contenido del vector, posición vigente.
            cout << *vector << endl;

            // Si en la posición vigente del vector contiene un 7
            // entonces se sale del ciclo 'access'.
            if(*vector == 7)
                break;
        }
```

---

[*] Entrada al programa principal: **entry**.

```
        Usos:
        (1)     entry int main(int argc, char **argv, char **env)
                {
                        cuerpo;
                }
        no se requieren cambios.

        (2)     entry void principal(void)
                {
                        cuerpo;
                }
        se cambia por:
                void main(void)
                {
                        cuerpo;
                }
```

```
        (3)     entry void main_program(void)
                {
                        cuerpo;
                }
        se cambia por:
                void main(void)
                {
                        cuerpo;
                }
        Para todos los casos se quita la palabra ’entry’ y se reemplaza el nombre
        de la función por ’main’.
        En Windows:
                entry WINAPI principal(HINSTANCE a, HINSTANCE b, LPTSTR c, int d)
                {
                        cuerpo;
                }
        Se cambia por:
                WINAPI WinMain(HINSTANCE a, HINSTANCE b, LPTSTR c, int d)
                {
                        cuerpo;
                }

Cuando ocurre esto en un mismo programa fuente:
        entry void entrada_principal(void)
        {
                cuerpoA;
        }

        void main(void)
        {
                cuerpoB;
        }

el compilador debe hacer:
        void main(void)
        {
                cuerpoA;
        }

        void _main_(void)
        {
                cuerpoB;
        }
En todo programa fuente tiene que haber una única función ’main’.
La palabra reservada ’entry’ aparece por primera en "The C programming language", Apéndice A, página 180, 1º
edición de 1978, edición inglesa. Escrito por Brian W. Kernighan y Dennis M. Ritchie.
Estaba reservada para uso futuro pero nunca se implementó en el compilador C.


[*] Salida del programa principal: exit.
Si hay una entrada (entry) pues también hay una salida (exit).
La palabra reservada ’exit’ no va acompañada de paréntesis porque existe la
función ’exit’ en la biblioteca de funciones estándar de C/C++ <stdlib.h> y <stdlib>

    Uso:
        // función de entrada
        entry void main(int argc, char **argv)
        {
                cuerpo;
    }

    // función de salida
    exit int main(int retval)
    {
            cuerpo;
    }

Ejemplo:
    // función de entrada
    entry int main(int argc, char **argv)
    {
            hace algo aquí...;

            // Llama a la función de salida con el valor 0
            exit 0;
    }

    // función de salida
    exit int xmain(int retval)
    {
            // Por ejemplo libera espacio de memoria
            hace algo aquí...;

            // Valor de retorno para el S.O.
            return retval;
    }
```

Código de arranque para la nueva implementación (ec++):

```
        int main(int argc, char **argv, char **env)
        {
                int retval_main, retval_exit;

                retval_main = emain(argc, argv, env);
                retval_exit = xmain(retval_main);

                return retval_exit;
        }
```

No está permitido usar simultáneamente ’entry’ y ’exit’ en una misma función.
Por ejemplo:

```
        // función de entrada y salida
        exit entry int main(int argc, char **argv)
        {
                cuerpo;
        }
```

El traductor ec++ avisaría con un mensaje de error que se está usando ’entry’ y ’exit’ sobre una misma función.

[*] Vinculación con otros lenguajes de programación mediante la palabra reservada ’extern’:
        Uso: extern <sequencia de caracteres>

        Conexión a C: extern "c"
        Conexión a C++: extern "c++"
        Conexión a C#: extern "c#"
        Conexión a Java: extern "java"
        Conexión a Vala: extern "vala"

[*] Punteros a memoria.
Las funciones on()/off() están orientadas para activar/desactivar punteros a memoria.

Poner sus declaraciones en <pointers.h>:

```
    // Activa el puntero ’p’ y devuelve true/false (activado/desactivado)
    bool on(void *p);

   // Desactiva el puntero ’p’ y devuelve true/false (activado/desactivado)
   bool off(void *p);
```

Ejemplo práctico:

```
        void *ptr;
        char *cptr;

        // los punteros están desactivados.
        off(ptr);
        off(cptr);

        ptr = "activame";   <-- ERROR !! el puntero está desactivado.
        ptr = 0xabcdef00;   <-- ERROR !! el puntero está desactivado.
        ptr = new void[1024];  <-- ERROR !! el puntero está desactivado.
        cptr = new char[1024];  <-- ERROR !! el puntero está desactivado.

        // los punteros están activados.
        on(ptr);
        on(cptr);

        ptr = new void[1024];  <-- OK !! el puntero está activado.
        cptr = new char[1024];  <-- OK !! el puntero está activado.
```

[*] Ejemplos teóricos.
    Ejemplo sobre ’access’: se cuenta con un vector numérico de 10 elementos y se recorre el mismo desde la posición 0
    hasta la posición 9. Cada elemento del vector es visualizado en pantalla separado por un espacio en blanco.
    La dirección del recorrido es indicada con la palabra reservada ’forward’. El acceso a cada miembro del vector es indicado por el operador indirección ’*’, también llamado operador de desreferencia.
    Se puede observar la ausencia de una variable índice para poder acceder a un elemento del vector.

```
    #include <iostream>
    using namespace std;

    main()
    {
        int vector[] = { 4, 33, 23, 1, 0, 7, 99, -6, -62, 19 };

        access(vector; forward)
        {
            // Muestra el contenido del vector, posición vigente
            cout << *vector << " " << endl;
        }
    }
```

Si se quiere recorrer el vector desde atrás hacia adelante tengo que indicar con la palabra reservada ’rewind’:

```
...
access(vector; rewind)
...
```

Y si se quiere recorrer el vector desde una posición arbitraria tengo que indicar así:

```
int n = 4;
...
access(vector[n], forward)
...
```

Otros casos posibles:

```
access(vector[3:5]; forward)           // recorrido hacia adelante sobre un rango de posiciones: 3, 4 y 5.
access(vector[5:3]; rewind)            // recorrido hacia atrás sobre un rango de posiciones: 5, 4 y 3.
access(vector[5],vector[3]; specific)  // recorrido específico en las posiciones indicadas: 5 y 3.
access(vector; random)                 // hace un recorrido aleatorio completo sobre el vector.
```

_____

Ultima actualización: 11:08 a.m. sábado, 30 de enero de 2021