

Propuesta realizada por Octulio Biletán.

Las nuevas palabras reservadas son: entry, exit, access, forward, rewind, specific, random, loop, until y repeat.

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

```
(1)    Usar:
        do
        {
            cuerpo;
        }loop;

(2)    Usar:
        loop
        {
            cuerpo;
        }
```

---

Ciclo do...until

```
(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.

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 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 el arreglo unidimensional 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 el arreglo unidimensional 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;
}

```

[\*] Vinculación con otros lenguajes de programación mediante la palabra reservada 'extern':

Uso: extern <secuencia de caracteres>

```

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

```

[\*] 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: 12:26 p.m. jueves, 21 de enero de 2021