Manual de usuario

Concha Vázquez Miguel
mconcha@ciencias.unam.mx

Flores Martínez Andrés
andresfm97@ciencias.unam.mx

Gladín García Ángel Iván
angelgladin@ciencias.unam.mx

Sánchez Pérez Pedro Juan Salvador
pedro_merolito@ciencias.unam.mx

Vázquez Salcedo Eduardo Eder
eder.vs@ciencias.unam.mx

14 de diciembre de 2018

Índice

1. Secciones de un programa

Los programas definidos estarán compuestos por dos secciones: declaraciones y funciones. En cualquier momento se permite comentar el código; los comentarios inician con `/*` y terminan con `*/`. Todo comentario deberá cerrarse antes de terminar el programa y pueden ser multilínea.

1.1. Declaraciones

1.1.1. Declaración de variables de tipos básicos y arreglos

Para las declaraciones de este tipo —que pueden no estar presentes en el programa— se podrán definir varias variables en una misma línea de la forma:

```
tipo [lista de identificadores separados por comas] identificador;
```

Podrán definirse variables numéricas de tipo enteras, decimales (punto flotante de simple y doble precisión) y constantes de carácter. Para las estructuras, consultar la siguiente subsección. Las palabras reservadas para los tipos que deberán preceder, respectivamente, a los identificadores escogidor por el programador en cada caso son:

- `int`
- `float`
- `double`
- `char`

En caso de definir un arreglo, sus dimensiones se deberán especificar con números enteros entre corchetes y el tipo de los elementos que son almacenados en este deberá preder al nombre del arreglo que se elija:

```
tipo identificador[x][y]...;
```

Finalmente, los nombres de los identificadores deberán obedecer la regla de que tengan a lo más treinta y dos caracteres. Deberán:

- Iniciar con una letra del abecedario (sea minúscula o mayúscula), o bien con un guión bajo.
- Después, le podrán serguir opcionalmente más letras en minúsculas o mayúsculas, guiones bajos y dígitos.

1.1.2. Declaración de estructuras

Es posible definir estructuras; basta con usar la palabra reservada `struct` y posteriormente colocar sus variables dentro de llaves, de la forma:

```
struct {
  declaraciones
} nombre_estructura;
```

1.2. Funciones

En la sección de funciones se declaran las funciones y en su cuerpo se colocan las sentencias, mismas que incluirán llamadas a otras funciones previamente declaradas. La forma de declarar una función es:

```
func tipo id (argumentos) {
  declaraciones
   sentencias
}
```

`func` es la palabra reservada para definir funciones y `id` sería el identificador que se elija para nombrar a la función, siguiendo las mismas reglas descritas antes para los nombres de variables y expresiones. Es fundamental definir adecuadamente el tipo de retorno de la función, mismo que puede ser de tipo `void`.

1.2.1. Argumentos

Los argumentos de una función pueden ser vacíos, puede ser uno solo o bien una lista. En caso de tener argumentos, cada uno deberá tener primero su tipo, el identificador e irán separados por comas:

```
func tipo id (tipo id, tipo id, ...) {
  declaraciones
   sentencias
}
```

En caso de utilizar un arreglo como argumento en la declaración de las funciones, se deberán describir por medio del uso de corchetes las dimensiones del arreglo esperado como parámetro cuando la función sea llamada. Como ejemplo:

```
func tipo id (tipo arr[][][][]...) {
  declaraciones
   sentencias
}
```

1.2.2. Declaraciones

La sección de declaraciones dentro de las funciones se lleva a cabo de la misma manera a la que fue descrita para la sección del programa previa a las funciones.

1.2.3. Sentencias

Las funciones deberán tener forzosamente al menos una sentencia en su cuerpo, aunque se ofrece la posibilidad de que sean tantas como se considere necesario. Hay varios tipos de sentencias permitidas:

- Estructuras de control condicionales:
  - `if` e `if-else`
    Podrán definirse condicionales de este tipo con la siguiente extructura:

    ```
    if (condición) sentencia [else sentencia];
    ```

    En donde los corchetes del final indican que lo que deberá ejecutarse en caso de no cumplirse la condición es opcional. Las sentencias descritas podrían ser como cualquiera de las que se describen en esta sección. Por su parte, las condiciones pueden ir entre paréntesis opcionalmente y pueden corresponder a:
    - Disyunción de dos condiciones: `condición || condición`.
    - Conjunción de dos condiciones: `condición && condición`.
    - Negación de una condición: `! condición`.
    - Constante verdadera `true` o falsa `false`.
    - Una relación entre dos expresiones, en donde la relación podría ser:
      - `<` (menor que)
      - `<=` (menor o igual que)
      - `>` (mayor que)
      - `>=` (mayor o igual que)
      - `!=` (distinto de)
      - `==` (igualdad)
  - `switch`: La estructura que deberá seguirse para declarar un `switch` es la siguiente:

    ```
    switch(expresion) {
     case: NUMERO sentencia
      ...
      ...
      ...
      default: sentencia
    }
    ```

    tanto los casos como el caso por omisión (predeterminado) son opcionales, esto es, pueden o no ester presentes.

- Estructuras de control de repetición (bucles):
  - `while` y `do while`: Para las estructuras de control de repetición de este tipo, se tiene una estructura muy similar:
    - while:

      ```
      while (condicion) {
       sentencia
      }
      ```

    - do while:

      ```
      do sentencia while (condicion);
      ```

  - `for`: Ahora, para los ciclos de tipo `for`, se deberán escribir como:

```
for (sentencia; condicion; sentencia) sentencia
```

- Asignaciones:

  Una asigación consta de una parte izquierda, el signo de `=` y una expresión en la parte derecha, para luego ir terminada con un punto y coma de la siguiente manera:

  izquierda = expresión;

  La parte de la izquierda podría hacer referencia a una variable previamente declarada —incluyendo a un arreglo que se indexa— o a un campo de una estructura que también hubo que ser declarada antes[1]. La expresión podría ser cualquier expresión aritmética de las descritas antes, una cadena, una constante numérica de tipo antera o decimal o bien el resultado de una llamada a función.

- *Prints*:

  Bastará con hacer uso de la palabra reservada `print` seguida de la expresión que se desee:

  print expresión

- `returns` y `breaks`: Los `breaks` deberán estar declarados dentro de algún ciclo para que su uso sea el correcto. Por otro lado, en caso de usar un `return` dentor de una función, deberá ser porque su tipo de retorno es `void` de acuerdo a como fue declarada. Finalmente, es posible retornar algún número, constante de cadena o expresión en caso de que el tipo deestas coincidan con el tipo de returno la función. Para los tres casos, la forma de escribir estas sentencias es como sigue:

```
return expresion
```

```
return;
```

```
break;
```

2. Consideraciones por parte del programador

Al programar, con tal de que no haya errores semánticos en los programas definidos, se deberá tener cuidado con:

- Los archivos dentro de los cuales se escriba el código fuente permiten cualquier nombre y extensión deseada; no se impusieron restricciones en este sentido.
- Haber declarado antes el identificador en caso de querer usarlo dentro de algun expresión.
- Al llamar funciones, los parámetros que se reciben deberán ser provistos en el mismo orden, con el mismo tipo y deberán ser tantos como se especificó al declarar la lista de argumentos para la función en cuestión.
- El tipo de la expresión de retorno de las funciones deberá coincidir con el tipo de retorno que fue especificado el declarar la función; si una función es de tipo void, no se podrán devolver valores. Además, en caso de que la función reciba arreglos como parámetros, deberán ser del mismo tipo y dimensiones de aquellos que fueron definidos como parte de los argumentos en la declaración.
- No se podrán definir variable ni parámetros del tipo `void`.

[1] En este caso se haría referencia como `id.id`.

- No existen los prototipos de funciones, lo que implica que para poder llamar a una función, esta deberá haber sido declarada con antelación.
- Todas las funciones deberán declararse antes —nunca después— de la función `main`; además, la función `main` siempre deberá declararse para que el programa sea válido.
- En los casos de indexar arreglos, se debe verificar que:
  - Se haga con un número o expresión entera.
  - Que el entero o expresión no sea negativa.
  - Que el entero o expresión no exceda el tamaño de la dimensión que fue declarada para el arreglo.
  - Cuando se utilicen `switches`, el programador deberá tener cuidado de que el valor numérico sea entero.
  - Siempre que se haga referencia a un campo de alguna estructura, este deberá haber sido declarado antes.