C++ según yo, pero en pedo

Introducción:

Los archivos de código fuente de C++ suelen tener extensión ".cpp".

Existen archivos "de encabezado", que también contienen código fuente, y suelen utilizarse en situaciones especiales, aunque su uso no es obligatorio. Su extensión suele ser ".h".

Expresiones:

Una expresión es una combinación válida de operandos y operadores de expresiones que produce un único resultado.

Un operando puede ser un literal (valor escrito en forma explícita), una variable, una constante, el valor de retorno de un subprograma o una subexpresión.

Un operador de expresiones es un caracter o una secuencia de caracteres que indica una acción a realizar con uno o más operandos.

Una expresión puede contener otras, estas son delimitadas con paréntesis "`(subexpresion)`".

Evaluar una expresión significa efectuar las operaciones que indican los operandos sobre los operadores, evaluando subexpresiones si las hubiera.

Al evaluar expresiones, se sigue un orden de precedencia. Este indica que primero se evalúan las subexpresiones, luego ciertos operadores aritméticos, luego los de comparación y booleanos, en términos generales. Además, evalúan de izquierda a derecha.

Instrucciones:

Una instrucción es una orden que se le da a la computadora.

Las instrucciones terminan en punto y coma (`;`).

Las instrucciones pueden ser agrupadas mediante llaves (`{instruccion1; instruccion2; instruccion3}`), formando un bloque de código.

Comentarios:

Líneas de código ignoradas por el compilador.

Una línea es comentada a partir de que en ella se encuentran los caracteres "`//`".

Una parte del código es comentada si está encerrada de esta forma: "`/*comentario*/`".

Palabras reservadas:

Son combinaciones de caracteres con fines específicos. Pueden formar parte de una instrucción o no.

Estas palabras no pueden ser usadas para nombrar elementos definidos por el programador.

Variables:

Una variable es un contenedor de datos.

Una variable posee un tipo de dato, un nombre o identificador y su contenido.

Declarar una variable es establecer su tipo de dato y su nombre.

Definir una variable es establecer su contenido.

Una variable puede ser declarada una sola vez, pero definida cuantas veces haga falta.

Sintaxis para tipos de datos primitivos y algunos no primitivos:

```
modificadores tipoDeDato identificador;          //Declaración
identificador operadorDeAsignacion expresion;    //Definición
modificadores tipoDeDato identificador = expresion;
//^^Declaración y definición
```

Tipos de datos primitivos:

Un tipo de dato es una forma de organizar la memoria.

Al declarar una variable, se le asigna un tipo de dato, es decir, la forma que deberá tener su contenido.

C++ según yo, pero en pedo

Los tipos de datos primitivos son aquellos que C++ ofrece por defecto. Algunos son `int`, `float`, `double`, `char` y `bool`. Estas son sus palabras reservadas correspondientes.

Los modificadores son palabras reservadas que cambian alguna característica del comportamiento de las variables.

Los modificadores de tipo de dato son:

`signed` y `unsigned`: para `char`, `int` y `double`, indican si soportan o no valores negativos.

`long` y `short`: para `char`, `int` y `double` (no todos son válidos), aumentan o disminuyen el rango de valores que pueden tomar.

Otro modificador es `const`, que impide que una variable sea definida más de una vez.

Pueden combinarse múltiples modificadores.

Subprogramas:

Un subprograma es un proceso que puede tener entradas, una salida "principal" o retorno y otras salidas "secundarias".

Declarar un subprograma es establecer el tipo de dato de su retorno, su nombre o identificador y sus entradas o parámetros (formales).

Definir un subprograma consiste en declararlo y establecer el procedimiento que lleva a cabo, un bloque de instrucciones.

Un subprograma puede declararse múltiples veces, pero solo puede ser definido una vez.

Múltiples subprogramas pueden compartir identificador y hasta tipo de dato si sus parámetros no coinciden. A esto se le llama sobrecarga de funciones (*function overload*).

Sintaxis:

```
modificadores tipoDeDato identificador (parametros); //Declaración
modificadores tipoDeDato identificador (parametros) {procedimiento}; //Declaración y definición
```

Los parámetros deben ser colocados entre los paréntesis como una lista separada con comas (`,`), especificando en cada uno sus modificadores, tipo de dato e identificador, como si de la declaración de una variable se tratara.

Llamar un subprograma es ejecutarlo, pasándole los parámetros que necesita y haciéndolo seguir su procedimiento.

Un subprograma puede ser llamado desde dentro de una expresión, siendo su valor de retorno un operando.

Sintaxis:

```
identificador(parametros);     //Llamada simple a un subprograma
variable = identificador(parametros);
/*              ^^Llamada a un subprograma, cuyo valor de retorno es almacenado en variable */
```

Un subprograma puede no retornar nada; si se busca este comportamiento, al declararlo se debe colocar la palabra reservada "`void`" en lugar de los modificadores y el tipo de dato. Un subprograma así no puede ser usado en expresiones.

Si un subprograma fue declarado de forma que retorne algo, esto se logra colocando una o más instrucciones "`return variable;`", donde "`return`" es una palabra reservada que indica la finalización del procedimiento y variable es el identificador de una variable del mismo tipo de dato que el especificado en la declaración/definición del subprograma.

`main()` es una función que es llamada automáticamente al ejecutar el programa.

Estructuras de control:

Las estructuras de control son formas de ejecutar instrucciones una o múltiples veces, dependiendo de ciertas condiciones.

Las estructura condicional es `if/else if/else`, la cual permite ejecutar bloques de código dependiendo del valor de verdad de una expresión.

Los bucles inexactos son `while` y `do/while`, y permiten ejecutar instrucciones una cantidad por lo general desconocida de veces, siempre y cuando una expresión sea verdadera.

El bucle exacto es `for`, el cual facilita la ejecución de instrucciones de forma estructurada, con más claridad en el código. Esto lo logra definiendo una variable de control una vez, evaluando una expresión que la involucra en cada iteración, ejecutando el bloque de instrucciones si la expresión es verdadera y modificando la variable de control. El primer, segundo y cuarto paso son realizados de forma clara al principio del bucle.

La sintaxis de estas estructuras está en el documento "C++ según yo".

Tipos de datos no primitivos:

Los tipos de datos no primitivos son aquellos definidos por programadores que usan C++.

También son conocidos como registros u objetos.

En C++, pueden ser creados de dos formas: mediante estructuras (`struct`) o mediante clases (`class`). Ambos son muy similares, los `struct` fueron heredados del predecesor de C++, C; mientras que las `class` son el método más popular de crear tipos de dato no primitivos. Me voy a enfocar en explicar los `struct`.

Sintaxis:

```
struct tipoDeDatoNoPrimitivo {propiedades y métodos} variables;
//^^Creación de un struct.
```

El tipo de dato no primitivo es el nombre bajo el cual identificaremos a este `struct`.

Las variables es una lista de variables de este tipo de dato, que son declaradas en este momento.

Tanto el tipo de dato como las variables son opcionales, pero siempre debe haber al menos uno de estos dos elementos al crear un struct.

Las propiedades son variables dentro del struct. Se declaran y definen de la misma forma que una variable común.

Los métodos son subprogramas dentro del struct. Se declaran y definen de la misma forma que un subprograma común.

Declarar una variable de un tipo de dato no primitivo puede hacerse en el struct (como ya fue mostrado) o como una variable de un tipo de dato primitivo.

Definir una variable de un tipo de dato no primitivo no siempre es sencillo. En algunos casos específicos, como `string`, se lo puede hacer como si fuera primitivo, pero en muchos otros consiste en definir cada una de sus propiedades manualmente.

Para acceder a una propiedad o método de un struct, se debe usar el operador punto (`.`). Así, se puede definir propiedades o métodos, así como recuperar sus valores o llamarlos, respectivamente.

Sintaxis:

```
variable.propiedad = expresion //Definir una propiedad
variable.metodo()              //Llamar a un método
variable.propiedad.propiedad
//^^Acceder a una propiedad de una propiedad (de tipo de dato no
//primitivo, a su vez)
```

Los `struct` soportan los constructores, métodos llamados automáticamente al declarar una variable de su tipo, que facilitan la definición.

Punteros:

Los punteros (pointers) son un tipo especial de variable que guardan la dirección de memoria de otra variable del mismo tipo de dato.

Para manipular punteros, existen dos operadores principales: dirección-de/address-of (`&`) y derreferencia/dereference (`*`).

`&variable` devuelve la dirección en que una variable está almacenada.

`*puntero` recupera el valor de la variable en la dirección almacenada.

C++ según yo, pero en pedo

Tener en cuenta que, si bien para declarar un puntero se usa el símbolo *, este no es el operador de derreferencia.

Sintaxis:

```
modificadores tipoDeDato * identificador;  //Declaración
identificador = &variable;                  //Definición
modificadores tipoDeDato * identificador = &variable;
//^^Declaración y definición
variable = *identificador;                 //Derreferenciación
```

Los punteros permiten el pasaje de variables por referencia a subprogramas. Para especificar que un cierto parámetro debe ser pasado por referencia, se puede:

Agregar el símbolo & (no es el operador dirección-de) entre el tipo de dato y el identificador del parámetro formal, o

Indicar que los parámetros formales son punteros, y al llamar al subprograma, pasarle punteros.

Arreglos:

Los arreglos son un espacio contiguo en la memoria, asignado para la ubicación de múltiples valores de un mismo tipo de dato.

Los arreglos se caracterizan por:

La dirección de memoria de su primer elemento (elemento 0). Esto puede ser almacenado en un puntero. Al pasar un arreglo como parámetro a un subprograma, en realidad pasamos este puntero.

Su cantidad de dimensiones, un número entero positivo.

La capacidad de cada dimensión, es decir, la cantidad de valores que puede almacenar en cada una. La capacidad total se define como el producto de las capacidades de cada dimensión.

Declarar un arreglo es determinar su tipo de dato e identificador.

Asignar memoria a un arreglo es determinar sus dimensiones y su capacidad. Esto se puede lograr gracias a subprogramas ya programados como *malloc*.

Definir un arreglo es dar valores a sus elementos.

Sintaxis:

```
modificadores tipoDeDato identificador[capacidad];
//^^Declaración y asignación de memoria de arreglo unidimensional
modificadores tipoDeDato identificador[capacidad1][capacidad2]^;
//^^Declaración y asignación de memoria de arreglo bidimensional
modificadores tipoDeDato identificador[];
//^^Declaración de arreglo unidimensional
modificadores tipoDeDato identificador[3] = {var1, var2, var3};
//^^Declaración, asignación de memoria y definición de arreglo
//unidimensional
modificadores tipoDeDato * identificador;
identificador = (tipoDeDato*) malloc(sizeof(tipoDeDato) * 10);
/*Esto último es la declaración de un puntero y la asignación de
memoria para 10 elementos mediante el subprograma malloc*/
```

Para acceder a los valores de un arreglo, se debe colocar las coordenadas de cada dimensión entre corchetes ([]).

Sintaxis:

```
identificador[expresion1] = expresion2;          //Definir el valor
//del elemento expresion1 del arreglo identificador como el valor
//de expresion2
variable = identificador[expresion1][expresion2];//Definir el valor
//de variable como el valor del elemento expresion2 del elemento
//expresion1 del arreglo identificador
```

Preprocesador:

El preprocesador es un programa llamado por el compilador antes de compilar un programa. El preprocesador ejecuta directivas (líneas empezadas con #), y permite darle más flexibilidad al código fuente.

La directiva #include permite dividir el código de un programa en diferentes archivos. Su funcionamiento está explicado en el archivo “C++ según yo”.