C – (minus)

Es un subconjunto restringido de C. Contiene:

- Enteros, arreglos de enteros, funciones.
- declaraciones locales y globales
- funciones recursivas (simples).
- Sentencias *if* y sentencia *while*
- La función main se debe declarar al último
- La ejecución comienza con una llamada a main.

Ejemplo

```
int fact( int x )
/* función factorial recursiva */
{ if (x > 1)
    return x * fact(x-1);
  else
    return 1;
}

void main( void )
{ int x;
  x = read();
  if (x > 0) write( fact(x) );
}
```

Gramática de C- (C minus)

1. *program* → *declaration-list*
2. *declaration-list* → *declaration-list declaration* | *declaration*
3. *declaration* → *var-declaration* | *fun-declaration*
4. *var-declaration* → *type-specifier* **ID** **;** | *type-specifier* **ID** **[** **NUM** **]** **;**
5. *type-specifier* → **int** | **void**
6. *fun-declaration* → *type-specifier* **ID** **(** *params* **)** *compound-stmt*
7. *params* → *param-list* | **void**
8. *param-list* → *param-list* **,** *param* | *param*
9. *param* → *type-specifier* **ID** | *type-specifier* **ID** **[** **]**
10. *compound-stmt* → **{** *local-declarations statement-list* **}**
11. *local-declarations* → *local-declarations var-declaration* | *empty*
12. *statement-list* → *statement-list statement* | *empty*
13. *statement* → *expression-stmt* | *compound-stmt* | *selection-stmt* | *iteration-stmt* | *return-stmt*
14. *expression-stmt* → *expression* **;** | **;**
15. *selection-stmt* → **if** **(** *expression* **)** *statement* | **if** **(** *expression* **)** *statement* **else** *statement*
16. *iteration-stmt* → **while** **(** *expression* **)** *statement*
17. *return-stmt* → **return** **;** | **return** *expression* **;**
18. *expression* → *var* **=** *expression* | *simple-expression*
19. *var* → **ID** | **ID** **[** *expression* **]**
20. *simple-expression* → *additive-expression relop additive-expression* | *additive-expression*
21. *relop* → **<=** | **<** | **>** | **>=** | **==** | **!=**
22. *additive-expression* → *additive-expression addop term* | *term*
23. *addop* → **+** | **-**
24. *term* → *term mulop factor* | *factor*
25. *mulop* → ***** | **/**
26. *factor* → **(** *expression* **)** | *var* | *call* | **NUM**
27. *call* → **ID** **(** *args* **)**
28. *args* → *arg-list* | *empty*
29. *arg-list* → *arg-list* **,** *expression* | *expression*

Proyecto

Dada la gramática de C– realice un programa que pase de código en C – a código de tres direcciones.

(Puntos extra para la calificación final hasta 10 puntos extra)

Dada la gramática de C- implemente un programa en C o C++ que almacene en una estructura de datos el código de tres direcciones y muéstrelo de una manera adecuada en pantalla

Ejemplo

Tomemos como ejemplo de entrada el código en C- mostrado al principio

El código de 3 direcciones será:

```
entry fact
    t1 = x > 1
    if false t1 goto L1
        begin_args
        t2 = x - 1
        param t2
        t3 = call fact, 1
        t4 = x * t3
        return t4
    Label L1
        return 1
entry main
    read x
    t5 = x > 0
    if false t5 goto L2
        begin_args
        param x
        t6 = call fact, 1
        write t6
    Label L2
    return
```

En este ejemplo se utilizó tabuladores para indicarnos el nivel de anidación, esto es solamente para poder verlo más claramente en comparación con el código, normalmente el código de tres direcciones va en la misma dirección (sin tabulador)

Pistas para el proyecto

Utilice un procedimiento a la gramática de atributos la cual, dada una regla sintáctica genere la cadena adecuada y vaya concatenando las reglas para al final obtener una cadena la cual será el código de tres direcciones, puede utilizar el operador "++" como concatenación.

Puede asumir que existe una función "`new_temp()`" la cual se encargara de dar un nuevo nombre temporal a una variable.

Puede asumir que existe una función "`new_label()`" la cual se encargara de dar un nuevo nombre a la etiqueta label.

Para saber dónde colar la etiqueta label puede utilizar una pila y hacer un push o pop para colocar la palabra.