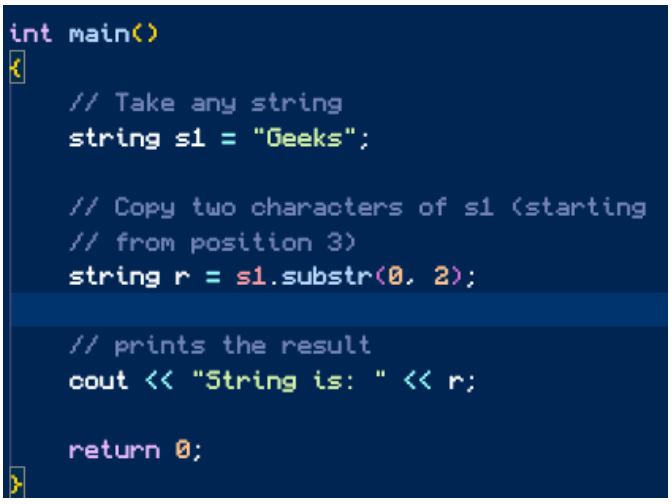
LAB 4

TOKEN

Bool isNumber: utilizzo questa funzione ausiliare per verificare se la stringa è un numero.
Per controllare mi basta un ciclo for per scorrere ogni carattere, appena un carattere non è una cifra restituisco false.

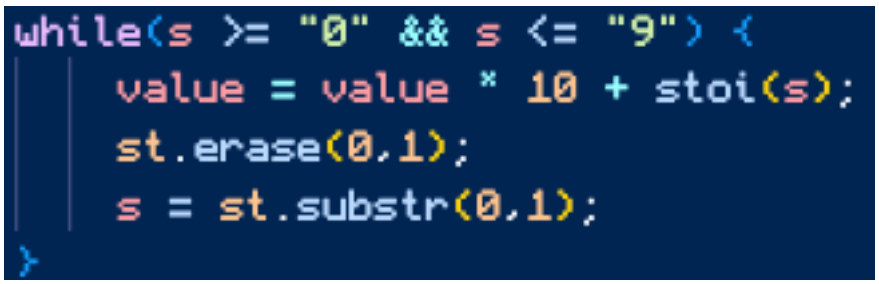
Bool nextToken: la funzione estrae il prossimo token della string st.
Il token sarebbe il tipo di carattere, SOMMA, ADDIZIONE, PARENTESI APERTA ecc.
Il primo controllo è quello di verificare se la stringa è vuota.
Devo eliminare gli spazi iniziali.
Per farlo estraggo il primo carattere tramite substr.
Il funzionamento è Il seguente:
string_name.substr(size_t pos, size_t len).
Facciamo un esempio :



In questo codice con s1.substr(0,2) otteniamo in output GE, 0 indica l'indice di partenza, quindi la pos. Il 2 indica invece quanti caratteri stampare.

Tornando alla nostra funzione iniziale a noi ci interessa solo il primo carattere, quindi avremo come parametri (0,1).
Ora dobbiamo eliminare tutti gli spazi, per farlo usiamo un ciclo while.
Per eliminare il carattere usiamo st.erase(0,1) . Significa di eliminare il primo carattere all'indice 0. Una volta eliminato dobbiamo riaggiornare il carattere usando s = st.substr(0,1).
Una volta eliminati tutti gli spazi controlliamo il primo carattere per capire che tipo di token abbiamo.
Avremo una serie di if else per verificare i vari casi.
Ad ogni caso associo il tok.k ovvero salvo il tipo del token che abbiamo trovato.
Se per esempio s == "(" allora nel tok.k associamo PARENTESI_APERTA.
Dopo aver identificato il tipo di token, dobbiamo rimuovere il carattere identificato dalla stringa st e aggiornare s per verificare se ci sono ulteriori spazi da eliminare.
Se il carattere successivo non è uno spazio allora lancio un errore in quanto per definizione la stringa deve avere uno spazio per ogni carattere (leggete con attenzione il testo di questo lab, lo dice all'inizio).
Ora abbiamo il caso della parentesi chiusa.
Se il primo carattere è una parentesi chiusa ")", impostiamo tok.k al valore PARENTESI_CHIUSA. Rimuoviamo la parentesi dalla stringa e controlliamo che il carattere successivo sia uno spazio o la stringa sia vuota. Se non lo è, solleviamo un'eccezione.
Ora ultimo caso in cui il carattere è un numero.
Controllo se il carattere è un numero richiamanado la funzione ausiliare isNumber spiegata sopra.
Viene inizializzata una variabile value a 0. Questa variabile verrà utilizzata per costruire il valore numerico.

Costruzione del numero:



Questo ciclo while continua finché il carattere corrente s è una cifra numerica.
value = value * 10 + stoi(s);: Converte il carattere s in un intero con stoi(s) e lo aggiunge a value. Moltiplica value per 10 prima di aggiungere la nuova

cifra per mantenere il valore corretto del numero (ad esempio, per costruire 123 da '1', '2', '3').
st.erase(0,1);: Rimuove il primo carattere dalla stringa st.
s = st.substr(0,1);: Aggiorna s con il nuovo primo carattere della stringa st.
Dopo aver costruito il numero, si verifica che il carattere successivo sia uno spazio o che la stringa sia vuota. Se non è così, viene sollevata un'eccezione con il messaggio "Lexical error".
Impostiamo il tipo del token a NUMERO e il valore del token a value.

STACK

Andiamo a modificare il file h copiando quello del lab 3:

```
// Implementa STACK
namespace stack{
    // tipo base
    typedef token Elem;

    const unsigned int BLOCKDIM = 10;

    typedef struct {
        //array dove sarono messi gli elementi
        Elem * data;
        //posizione del ultimo elemento
        unsigned int size;
        //lunghezza dell'array
        unsigned int maxsize;
    } Stack;
```

L'implementazione delle funzioni è la stessa, guardare spiegazione lab3.

ARITEXPR

Creiamo una pila vuota tramite stack::createEmpty.
Inizializziamo 4 variabili:

ST è una copia della stringa di input st.
Syn è il messaggio di errore sintattico.
el e op sono variabili di tipo Elem (alias di token).

Usiamo un ciclo per estrarre I Token, il ciclo while continua finché ci sono token da estrarre dalla stringa ST.
Se il token è una parentesi aperta (, un numero, +, - o *, lo aggiungiamo alla pila.

Quando incontriamo una parentesi chiusa), eseguiamo i seguenti passaggi:

- Estraiamo il secondo operando dalla pila.
- Verifichiamo che sia un numero (el.k == 2).
- Estraiamo l'operatore dalla pila.
- Estraiamo il primo operando dalla pila.
- Verifichiamo che sia un numero (el.k == 2).
- Estraiamo la parentesi aperta dalla pila e verifichiamo che sia ((el.k == 0).
- Eseguiamo l'operazione aritmetica corrispondente (+, -, *) e mettiamo il risultato nella pila come un nuovo token di tipo NUMERO.

Dopo aver processato tutti i token, l'ultimo elemento rimanente nella pila dovrebbe essere un unico numero.
Se la pila non è vuota o l'ultimo elemento non è un numero, solleviamo un'eccezione di errore sintattico.
Restituiamo il valore del token finale.