

```

1  /* /**/
2  * ASSIGNMENT: LL(1) PARSER
3  * AUTHORS: BIJAY KHATRI & DEEPAK VERMA
4  * ROLL: 12/CS/45 AND 12/CS/46
5  * /**/ */
6
7  #include<stdio>
8  #include<iostream>
9  #include<stdlib>
10 #include<cstring>
11 #define MACRO if(isalpha(a[i]) || a[i]=='_') \
12     { \
13         while(1) \
14         { \
15             j=0; \
16             while(a[i]!=' ')\
17                 i++; \
18             while(a[i]!=',' && a[i]!=';' && a[i]!='[' && a[i]!='=' && a[i]!='(' && a[i]!=' ' && a[i]!='{') \
19                 { \
20                     var[var_count].ar[j]=a[i]; \
21                     i++; \
22                     j++; \
23                 } \
24             var[var_count].ar[j++]='\0'; \
25             var[var_count].type=typ; \
26             if(a[i]!=' ')\
27                 {\
28                     while(a[i]!=' ')\
29                         i++; \
30                     i--; \
31                 } \
32             while(a[i]!='[') \
33             { \
34                 i++; \
35                 while(a[i]!='']') \
36                     i++; \
37                 i++; \
38             } \
39             if(a[i]=='(')\
40             { \
41                 i++; \
42                 while(a[i]!='')\
43                     i++; \
44             } \
45             if(a[i]=='=') \
46             { \
47                 i++; \
48                 if(a[i]!='{')\
49                 { \
50                     i++; \
51                     while(a[i]!='}')\
52                         i++; \
53                     if(a[i]=='')\
54                     { \
55                         i++; \
56                         while(a[i]!='')\
57                             i++; \
58                     } \
59                     if(a[i]=='')\
60                     { \
61                         i++; \
62                         while(a[i]!='}')\
63                             i++; \
64                     } \
65                     if(a[i]!='')\
66                     { \
67                         i++; \
68                         while(a[i]!='')\
69                             i++; \
70                     } \
71                 } \
72             } \
73             if(a[i]!=',' || a[i]!=';') \
74             { \
75                 break; \
76                 i=i++; \
77                 var_count++; \
78             } \
79         } \
80     } \
81
82 using namespace std;
83 struct lex
84 {
85     int countt;
86     int arr[500];
87 };
88 struct variable
89 {
90     int type;

```

```

78  int countt;
79  char ar[32];
80  int line[500];
81  bool error;
82  };
83  int main()
84  {
85      int k,total_size,var_count,typ,ln,start;
86      FILE *fptr;
87      char a[100000],str[100],c;
88      char key[30][10]={"if","else","switch","break","for","+","-","*","/","%","<","<=",">",">=","==","!=","&&","||","=","++","--","{","}",";",";",
"case","goto","continue","while","do"};
89      int i,j,line_no=0;
90      lex b[100];
91      variable var[500];
92      i=0;
93      fptr = fopen("assignment7(L1).cpp", "r");
94      if (fptr == NULL)
95      {
96          printf("Cannot open file \n");
97          exit(0);
98      }
99      c = fgetc(fptr);
100     while (c != EOF)
101     {
102         a[i++]=c;
103         c = fgetc(fptr);
104     }
105     total_size=i;
106     for(k=0;k<=100;k++)
107         b[k].countt=0;
108     for(j=0;j<=i;j++)
109     {
110         if(a[j]=='i' && a[j+1]=='f')
111         {
112             b[0].countt++;
113             b[0].arr[b[0].countt]=line_no;
114             j=j+1;
115         }
116         else if(a[j]=='d' && a[j+1]=='o')
117         {
118             b[29].countt++;
119             b[29].arr[b[29].countt]=line_no;
120             j=j+1;
121         }
122     }
123     else if(a[j]=='e' && a[j+1]=='l' && a[j+2]=='s' && a[j+3]=='e')
124     {
125         {
126             b[1].countt++;
127             b[1].arr[b[1].countt]=line_no;
128             j=j+3;
129         }
130     }
131     else if(a[j]=='w' && a[j+1]=='h' && a[j+2]=='i' && a[j+3]=='l' && a[j+4]=='e')
132     {
133         b[28].countt++;
134         b[28].arr[b[28].countt]=line_no;
135         j=j+4;
136     }
137     }
138     else if(a[j]=='c' && a[j+1]=='a' && a[j+2]=='s' && a[j+3]=='e')
139     {
140         b[25].countt++;
141         b[25].arr[b[25].countt]=line_no;
142         j=j+3;
143     }
144     else if(a[j]=='g' && a[j+1]=='o' && a[j+2]=='t' && a[j+3]=='o')
145     {
146         b[26].countt++;
147         b[26].arr[b[26].countt]=line_no;
148         j=j+3;
149     }
150     else if(a[j]=='s' && a[j+1]=='w' && a[j+2]=='i' && a[j+3]=='t' && a[j+4]=='c' && a[j+5]=='h')//switch
151     {b[2].countt++;
152     b[2].arr[b[2].countt]=line_no;
153     j=j+5;

```

```

154     }
155     else if(a[j]=='c' && a[j+1]=='o' && a[j+2]=='n' && a[j+3]=='t' && a[j+4]=='i' && a[j+5]=='n' && a[j+6]=='u' && a[j+7]=='e' )
156     ///CONTINUE
157     {b[27].countt++;
158     b[27].arr[b[27].countt]=line_no;
159     j=j+7;
160     }
161     else if(a[j]=='b' && a[j+1]=='r' && a[j+2]=='e' && a[j+3]=='a' && a[j+4]=='k')/// break
162     {
163     {b[3].countt++;
164     b[3].arr[b[3].countt]=line_no;
165     j=j+4;
166     }
167     }
168     else if(a[j]=='f' && a[j+1]=='o' && a[j+2]=='r')///for
169     {b[4].countt++;
170     b[4].arr[b[4].countt]=line_no;
171     j=j+2;
172     }
173     else if(a[j]=='+')
174     { if(a[++j]=='+')
175     {b[19].countt++;
176     b[19].arr[b[19].countt]=line_no;
177     }
178     }
179     else {b[5].countt++;
180     b[5].arr[b[5].countt]=line_no;
181     }
182     }
183     else if(a[j]=='-')
184     { if(a[++j]=='-')
185     {b[20].countt++;
186     b[20].arr[b[20].countt]=line_no;
187     }
188     }
189     }
190     else if(a[j]=='*')
191     {b[7].countt++;
192     b[7].arr[b[7].countt]=line_no;
193     }
194     }
195     else if(a[j]=='/')
196     {b[8].countt++;
197     b[8].arr[b[8].countt]=line_no;
198     }
199     }
200     else if(a[j]=='%')
201     {b[9].countt++;
202     b[9].arr[b[9].countt]=line_no;
203     }
204     }
205     else if(a[j]=='*')
206     {b[5].countt++;
207     b[5].arr[b[5].countt]=line_no;
208     }
209     }
210     else if(a[j]=='<')
211     {
212     if(a[++j]=='=')
213     {b[11].countt++;
214     b[11].arr[b[11].countt]=line_no;
215     }
216     }
217     }
218     else {
219     b[10].countt++;
220     b[10].arr[b[10].countt]=line_no;
221     }
222     }
223     }
224     else if(a[j]=='>')
225     {
226     if(a[++j]=='=')
227     {b[13].countt++;
228     b[13].arr[b[13].countt]=line_no;
229     }
230     }
231     }
232     }
233     }
234     }
235     }
236     }
237     }
238     }
239     }
240     }
241     }
242     }
243     }
244     }
245     }
246     }
247     }
248     }
249     }
250     }
251     }
252     }
253     }
254     }
255     }
256     }
257     }
258     }
259     }
260     }
261     }
262     }
263     }
264     }
265     }
266     }
267     }
268     }
269     }
270     }
271     }
272     }
273     }
274     }
275     }
276     }
277     }
278     }
279     }
280     }
281     }
282     }
283     }
284     }
285     }
286     }
287     }
288     }
289     }
290     }
291     }
292     }
293     }
294     }
295     }
296     }
297     }
298     }
299     }
300     }
301     }
302     }
303     }
304     }
305     }
306     }
307     }
308     }
309     }
310     }
311     }
312     }
313     }
314     }
315     }
316     }
317     }
318     }
319     }
320     }
321     }
322     }
323     }
324     }
325     }
326     }
327     }
328     }
329     }
330     }
331     }
332     }
333     }
334     }
335     }
336     }
337     }
338     }
339     }
340     }
341     }
342     }
343     }
344     }
345     }
346     }
347     }
348     }
349     }
350     }
351     }
352     }
353     }
354     }
355     }
356     }
357     }
358     }
359     }
360     }
361     }
362     }
363     }
364     }
365     }
366     }
367     }
368     }
369     }
370     }
371     }
372     }
373     }
374     }
375     }
376     }
377     }
378     }
379     }
380     }
381     }
382     }
383     }
384     }
385     }
386     }
387     }
388     }
389     }
390     }
391     }
392     }
393     }
394     }
395     }
396     }
397     }
398     }
399     }
400     }
401     }
402     }
403     }
404     }
405     }
406     }
407     }
408     }
409     }
410     }
411     }
412     }
413     }
414     }
415     }
416     }
417     }
418     }
419     }
420     }
421     }
422     }
423     }
424     }
425     }
426     }
427     }
428     }
429     }
430     }
431     }
432     }
433     }
434     }
435     }
436     }
437     }
438     }
439     }
440     }
441     }
442     }
443     }
444     }
445     }
446     }
447     }
448     }
449     }
450     }
451     }
452     }
453     }
454     }
455     }
456     }
457     }
458     }
459     }
460     }
461     }
462     }
463     }
464     }
465     }
466     }
467     }
468     }
469     }
470     }
471     }
472     }
473     }
474     }
475     }
476     }
477     }
478     }
479     }
480     }
481     }
482     }
483     }
484     }
485     }
486     }
487     }
488     }
489     }
490     }
491     }
492     }
493     }
494     }
495     }
496     }
497     }
498     }
499     }
500     }
501     }
502     }
503     }
504     }
505     }
506     }
507     }
508     }
509     }
510     }
511     }
512     }
513     }
514     }
515     }
516     }
517     }
518     }
519     }
520     }
521     }
522     }
523     }
524     }
525     }
526     }
527     }
528     }
529     }
530     }
531     }
532     }
533     }
534     }
535     }
536     }
537     }
538     }
539     }
540     }
541     }
542     }
543     }
544     }
545     }
546     }
547     }
548     }
549     }
550     }
551     }
552     }
553     }
554     }
555     }
556     }
557     }
558     }
559     }
560     }
561     }
562     }
563     }
564     }
565     }
566     }
567     }
568     }
569     }
570     }
571     }
572     }
573     }
574     }
575     }
576     }
577     }
578     }
579     }
580     }
581     }
582     }
583     }
584     }
585     }
586     }
587     }
588     }
589     }
590     }
591     }
592     }
593     }
594     }
595     }
596     }
597     }
598     }
599     }
600     }
601     }
602     }
603     }
604     }
605     }
606     }
607     }
608     }
609     }
610     }
611     }
612     }
613     }
614     }
615     }
616     }
617     }
618     }
619     }
620     }
621     }
622     }
623     }
624     }
625     }
626     }
627     }
628     }
629     }
630     }
631     }
632     }
633     }
634     }
635     }
636     }
637     }
638     }
639     }
640     }
641     }
642     }
643     }
644     }
645     }
646     }
647     }
648     }
649     }
650     }
651     }
652     }
653     }
654     }
655     }
656     }
657     }
658     }
659     }
660     }
661     }
662     }
663     }
664     }
665     }
666     }
667     }
668     }
669     }
670     }
671     }
672     }
673     }
674     }
675     }
676     }
677     }
678     }
679     }
680     }
681     }
682     }
683     }
684     }
685     }
686     }
687     }
688     }
689     }
690     }
691     }
692     }
693     }
694     }
695     }
696     }
697     }
698     }
699     }
700     }
701     }
702     }
703     }
704     }
705     }
706     }
707     }
708     }
709     }
710     }
711     }
712     }
713     }
714     }
715     }
716     }
717     }
718     }
719     }
720     }
721     }
722     }
723     }
724     }
725     }
726     }
727     }
728     }
729     }
730     }
731     }
732     }
733     }
734     }
735     }
736     }
737     }
738     }
739     }
740     }
741     }
742     }
743     }
744     }
745     }
746     }
747     }
748     }
749     }
750     }
751     }
752     }
753     }
754     }
755     }
756     }
757     }
758     }
759     }
760     }
761     }
762     }
763     }
764     }
765     }
766     }
767     }
768     }
769     }
770     }
771     }
772     }
773     }
774     }
775     }
776     }
777     }
778     }
779     }
780     }
781     }
782     }
783     }
784     }
785     }
786     }
787     }
788     }
789     }
790     }
791     }
792     }
793     }
794     }
795     }
796     }
797     }
798     }
799     }
800     }
801     }
802     }
803     }
804     }
805     }
806     }
807     }
808     }
809     }
810     }
811     }
812     }
813     }
814     }
815     }
816     }
817     }
818     }
819     }
820     }
821     }
822     }
823     }
824     }
825     }
826     }
827     }
828     }
829     }
830     }
831     }
832     }
833     }
834     }
835     }
836     }
837     }
838     }
839     }
840     }
841     }
842     }
843     }
844     }
845     }
846     }
847     }
848     }
849     }
850     }
851     }
852     }
853     }
854     }
855     }
856     }
857     }
858     }
859     }
860     }
861     }
862     }
863     }
864     }
865     }
866     }
867     }
868     }
869     }
870     }
871     }
872     }
873     }
874     }
875     }
876     }
877     }
878     }
879     }
880     }
881     }
882     }
883     }
884     }
885     }
886     }
887     }
888     }
889     }
890     }
891     }
892     }
893     }
894     }
895     }
896     }
897     }
898     }
899     }
900     }
901     }
902     }
903     }
904     }
905     }
906     }
907     }
908     }
909     }
910     }
911     }
912     }
913     }
914     }
915     }
916     }
917     }
918     }
919     }
920     }
921     }
922     }
923     }
924     }
925     }
926     }
927     }
928     }
929     }
930     }
931     }
932     }
933     }
934     }
935     }
936     }
937     }
938     }
939     }
940     }
941     }
942     }
943     }
944     }
945     }
946     }
947     }
948     }
949     }
950     }
951     }
952     }
953     }
954     }
955     }
956     }
957     }
958     }
959     }
960     }
961     }
962     }
963     }
964     }
965     }
966     }
967     }
968     }
969     }
970     }
971     }
972     }
973     }
974     }
975     }
976     }
977     }
978     }
979     }
980     }
981     }
982     }
983     }
984     }
985     }
986     }
987     }
988     }
989     }
990     }
991     }
992     }
993     }
994     }
995     }
996     }
997     }
998     }
999     }
1000    }

```

```

230         {b[14].countt++;
231           b[14].arr[b[14].countt]=line_no;
232         }
233     else
234     {b[18].countt++;
235       b[18].arr[b[18].countt]=line_no;
236     }
237 }
238 else if(a[j]=='!' && a[++j]== '=')
239   {b[15].countt++;
240    b[15].arr[b[15].countt]=line_no;
241   }
242 else if(a[j]=='|' && a[++j]=='|')
243   {b[17].countt++;
244    b[17].arr[b[17].countt]=line_no;
245   }
246 else if(a[j]=='&' && a[++j]=='&')
247   {b[16].countt++;
248    b[16].arr[b[16].countt]=line_no;
249   }
250 else if(a[j]== ',')
251   {b[23].countt++;
252    b[23].arr[b[23].countt]=line_no;
253   }
254 else if(a[j]== '}')
255   {
256     b[22].countt++;
257     b[22].arr[b[22].countt]=line_no;
258   }
259 else if(a[j]='{')
260   {
261     b[21].countt++;
262     b[21].arr[b[21].countt]=line_no;
263   }
264 else if(a[j]== ';')
265   {
266     b[24].countt++;
267     b[24].arr[b[24].countt]=line_no;
268   }
269 else if (a[j]=="")
270 { ++j;
271   while(a[j]!='')
272     ++j;
273 }
274 else if (a[j]=='[')
275 { ++j;
276   while(a[j]!=']')
277     ++j;
278 }
279 }
280 else if(a[j]=='\n')
281   line_no++;
282 else continue;
283 }
284 printf("\n\nTOKEN\t\t\t TOKEN TYPE\n\n\n");
285 for(j=0;j<30;j++)
286 {
287   for(i=0;key[j][i]!='\0';i++)
288   {
289     printf("%c",key[j][i]);
290   }
291   if((j>=0 && j<=4) || (j>=25 && j<=29))
292     printf("\t\t\t KEYWORD \n");
293   else printf("\t\t\t OPERATORS \n");
294 }
295
296 var_count=0;
297 for(i=0;i<total_size;i++)
298 {
299   if(a[i]=='i' && a[i+1]=='n' && a[i+2]=='t' && a[i+3]==' ')
300   { i=i+4;
301     typ=1;
302     MACRO;
303   }
304   else if(a[i]=='f' && a[i+1]=='l' && a[i+2]=='o' && a[i+3]=='a' && a[i+4]=='t' && a[i+5]==' ')
305   { i=i+6;
306     typ=2;
```

```

307     MACRO;
308 }
309 else if(a[i]=='c' && a[i+1]=='h' && a[i+2]=='a' && a[i+3]=='r' && a[i+4]==' ')
310 { i=i+5;
311     typ=3;
312     MACRO;
313 }
314 else if(a[i]=='d' && a[i+1]=='o' && a[i+2]=='u' && a[i+3]=='b' && a[i+4]=='l' && a[i+5]=='e' && a[i+6]==' ')
315 { i=i+7;
316     typ=4;
317     MACRO;
318 }
319 }
320 for(j=0;j<=var_count;j++)
321 { line_no=1;
322     ln=0;
323     for(i=0;i<total_size;i++)
324     { start=i;
325         for(k=0;var[j].ar[k]!='\0';k++)
326         { if(i-1>=0 && !isalpha(a[i-1]))
327             {
328                 while(var[j].ar[k]==a[i])
329                 {
330                     i++;
331                     k++;
332                     if(var[j].ar[k]=='\0' || a[i]==' ')
333                         break;
334                 }
335                 if(var[j].ar[k]!='\0' && !isalpha(a[i]))
336                 {
337                     var[j].countt++;
338                     var[j].line[ln++]=line_no;
339                     break;
340                 }
341                 if(a[i]=='\n')
342                 {
343                     line_no++;
344                     break;
345                 }
346                 if(a[i]==' ')
347                 {
348                     break;
349                 }
350                 if(var[j].ar[k]!='\0')
351                 {
352                     i=start;
353                     break;
354                 }
355             }
356         }
357     }
358 }
359 for(j=0;j<var_count;j++)
360 {
361     for(i=0;var[j].ar[i]!='\0';i++)
362     {
363         if(!(isalpha(var[j].ar[0]) || var[j].ar[0]=='_'))
364             var[j].error=true;
365         if(!(isalnum(var[j].ar[i]) || var[j].ar[i]=='_'))
366             var[j].error=true;
367     }
368 }
369 /// printf("\n\nIDENTIFIER\t\t\t\n");
370 for(j=0;j<=var_count;j++)
371 { // printf("\n");
372     { if(var[j].error)
373         printf("ERROR PRESENT IN SYMBOL\n");
374         for(i=0;var[j].ar[i]!='\0';i++)
375             printf("%c",var[j].ar[i]);
376         printf("\t\t\tIDENTIFIER\n");
377     }
378 }
379 fclose(fp);
380 return 0;
381 }
382
383

```

```

384 /*
385 //////////////////////////////////////
386 TOKEN      TOKEN TYPE
387 -----
388 if          KEYWORD
389 else        KEYWORD
390 switch      KEYWORD
391 break       KEYWORD
392 for         KEYWORD
393 +           OPERATORS
394 -           OPERATORS
395 *           OPERATORS
396 /           OPERATORS
397 %           OPERATORS
398 <           OPERATORS
399 <=          OPERATORS
400 >           OPERATORS
401 >=          OPERATORS
402 ==          OPERATORS
403 !=          OPERATORS
404 &&          OPERATORS
405 ||          OPERATORS
406 =           OPERATORS
407 ++          OPERATORS
408 --          OPERATORS
409 {           OPERATORS
410 }           OPERATORS
411 ,           OPERATORS
412 ;           OPERATORS
413 case        KEYWORD
414 goto        KEYWORD
415 continue    KEYWORD
416 while       KEYWORD
417 do          KEYWORD
418 k           IDENTIFIER
419 total_size  IDENTIFIER
420 var_count   IDENTIFIER
421 typ         IDENTIFIER
422 ln          IDENTIFIER
423 a           IDENTIFIER
424 str         IDENTIFIER
425 i           IDENTIFIER
426 j           IDENTIFIER
427 line_no     IDENTIFIER
428
429 */
430
431

```