**Compiler**

과제 8 : Making AST Using Bison

이름 : 제정륭

학번 : 201224537

att.l

|  |
| --- |
| %{  #include "ast.h"  #include "y.tab.h"  #include <stdlib.h>  %}  %%  [0-9]+ { yylval.iVal = atoi(yytext); return NUM; }  [ \t] ;  \n return 0;  "+" return '+';  "-" return '-';  "\*" return '\*';  "/" return '/';  "(" return '(';  ")" return ')';  . { printf("error near %s\n", yytext); exit(-1); }  %%  int yywrap() {  return 1;  } |

att.y

|  |
| --- |
| %{ #include <stdio.h>  #include <ctype.h>  #include "ast.h"  int yyerror(const char \*msg), yylex();  %}  %union {  Tree pVal;  int iVal;  char \*sVal;  }  %type <pVal> Exp  %token <iVal> NUM  %left '-' '+'  %left '\*' '/'  %nonassoc UNARYMINUS  %%  Line : Exp { printtree($1,0); }  ;  Exp : Exp '+' Exp { $$ = mknode(PLUS, 0, NULL, $1);  $1->bro = $3; }  | Exp '-' Exp { $$ = mknode(MINUS, 0, NULL, $1);  $1->bro = $3;}  | Exp '\*' Exp { $$ = mknode(MUL, 0, NULL, $1);  $1->bro = $3;}  | Exp '/' Exp { $$ = mknode(DIVIDE, 0, NULL, $1);  $1->bro = $3;}  | '-' Exp %prec UNARYMINUS { $$ = mknode(UMINUS, 0, NULL, $2); }  | '(' Exp ')' { $$ = $2; }  | NUM { $$ = mkleaf(INT, $1); }  ;  %%  int main() { yyparse(); return 0; }  int yyerror(const char \*msg) { fputs(msg, stderr); }  /\*  int yylex() { int c = getchar();  return (c == '\n')? 0: (isdigit(c))? yylval=c-'0', NUMBER: c;  }  \*/ |

* Unary minus는 %prec를 이용해서 우선순위를 UNARYMINUS로 정의한 만큼 상승시켜서 구현했습니다.

ast.h

|  |
| --- |
| #ifndef \_\_AST\_H\_\_  #define \_\_AST\_H\_\_  typedef enum { PLUS, MINUS, MUL, DIVIDE, UMINUS, INT, CHAR } NKind;  typedef struct \_node {  NKind kind;  int depth;  union {  int ival;  char \*sval;  };  struct \_node \*bro, \*son;  } \*Tree;  Tree mkleaf(NKind kind, int val);  Tree mknode(NKind kind, int val, Tree bro, Tree son);  void printtree(Tree root, int depth);  #endif |

* 생성된 tree를 출력하는 함수인 printtree를 선언했습니다.

ast.c

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include "ast.h"  void printtree(Tree root,int depth) {  int i;  if(root == NULL)  return;  switch(root->kind){  case PLUS :  for(i=0; i<depth; ++i)  printf("\t");  printf("Op(+)\n");  printtree(root->son,depth+1);  printtree(root->bro,depth);  break;  case MINUS :  for(i=0; i<depth; ++i)  printf("\t");  printf("Op(-)\n");  printtree(root->son,depth+1);  printtree(root->bro,depth);  break;  case MUL :  for(i=0; i<depth; ++i)  printf("\t");  printf("Op(\*)\n");  printtree(root->son,depth+1);  printtree(root->bro,depth);  break;  case DIVIDE :  for(i=0; i<depth; ++i)  printf("\t");  printf("Op(/)\n");  printtree(root->son,depth+1);  printtree(root->bro,depth);  break;  case UMINUS :  for(i=0; i<depth; ++i)  printf("\t");  printf("Uop(-)\n");  printtree(root->son,depth+1);  printtree(root->bro,depth);  break;  case INT :  for(i=0; i<depth; ++i)  printf("\t");  printf("Int(%d)\n",root->ival);  printtree(root->son,depth+1);  printtree(root->bro,depth);  break;  case CHAR :  break;  }  }  Tree mkleaf(NKind kind, int val) {  Tree t = (Tree)malloc(sizeof \*t);  t->kind = kind;  t->ival = val;  t->bro = t->son = NULL;  return t;  }  Tree mknode(NKind kind, int val, Tree bro, Tree son) {  Tree t = (Tree)malloc(sizeof \*t);  t->kind = kind;  t->ival = val;  t->bro = bro;  t->son = son;  return t;  } |

* 트리는 노드의 속성에 따라서 recursive하게 출력하도록 만들었습니다.
* Tree의 depth는 argument로 recursive하게 반복적으로 전달해서 해당 노드의 depth만큼 들여쓰기하도록 만들었습니다.