p3:

a) program

Declarations: [int a, b, c; / double d;] → build symbol table

begin

Statements:
- b = 3*(5+2);
- c = (3+4)*5;
- print b;
- print c;
- print (2+3)*7+2^3;

end

```
main()
{ read a word;
  if (word == "program")
    [ Call declarations();
      Call statements(); ]
  else
    error; exit(1);
}
```

```
declarations() — recursive
{ read a word;
  if (word == "begin")
    [ exit(1); ]
  else ("int" or "double")
    Call declaration(); ← word (int/dou
    Call declarations();
}
```

ret.

build sym tab

int/double

```
declaration(word)
{ read id;
  store it in symbol_table[index].id;   (global or local)
  store word in symbol_table[index].type;
  index++;
  loop until ; meets;
}
```

i.e., [if ①, repeat / else ; — exit(1);] (read a char)

Symbol_table[ ] — <char, string, in

	id	type	value
0	a	int	
1	b	int	21
2	c	int	35
⋮	d	double	
		⋮	

```
struct node
{ char id;
  string type;
  int val;
};
struct node symbol_tab[
```

symbol_table[index].id = a;
symbol_table[index].type = "int";
symbol_table[index].val = 21;

↓

```
— statements() — recursive
{ read a word;
  if (word == "end")
      exit(1);
  else
      call statement(word);
  call statements();
}
```

rec

```
— statement(word)
{ if (word == "print")
      call print_st();
  else (id)
      call Assign_st(word);
}
```

id

```
— print_st()
{ read a char (id);
  if (id is 'a'~'z')
      search symbol_tab for id
          and get index;
      if (found)
          display symbol_tab[index].val;
      else
          semantic error;
  else (expression case)
      get back one position;
      call Exp() and display result;
}
```

extract val. from sym_tab.

```
— Assign_st(word)
{ read a word;
  if (word == "=")
      int temp = Exp();
      search symbol_tab for word
          and get index;
      if (found)
          symbol_tab[index].val = temp
      else
          semantic error;
  else
      syntax error.
```

enter val. to sym_tab