http://japaneseemoticons.me/all-japanese-emoticons/
http://textsmili.es/?cr=bW91dGh%2Bdy5udy5pZV9leWVzfncubzEuNHdfZWFyc34xNC0xNQ%3D%3D

```
"ゝ"                         { tokenChar(','); return MK_COMMA; }
"ʖ"                          { tokenChar(';'); return MK_SEMICOLON; }
"ↄ"                          { tokenChar(':'); return MK_COLON; }

"□"                          { tokenChar(')'); return MK_RPAREN; }
"□"                          { tokenChar('('); return MK_LPAREN; }
"□"                          { tokenChar('['); return MK_LB;}
"□"                          { tokenChar(']'); return MK_RB; }

" ┏( ^ 0 ^ )┛ "              { tokenChar('+'); return OP_ADD; }
" ┗( ^ 0 ^ )┓ "              { tokenChar('-'); return OP_SUB; }
" ┗|･.･┏|          "                 { tokenChar('*'); return OP_MUL; }
" ┗|･.･┏| "                  { tokenChar('/'); return OP_DIV; }
" ┏|*° -° |┛ "               { token(mod); return OP_MOD; }

"☜(゜∀゜☜)"                  { token(:=); return OP_ASSIGN; }
"(‘▿’ʃʃ"                     { tokenChar('<'); return OP_LT; }
"□ | − ف − |—∈"                     { token(<=); return OP_LE; }
"□ | − ف − | □"                     { token(<>); return OP_NE; }
"3-| - ف - |?"               { token(>=); return OP_GE; }
"ʅʅ’▿’)"                     { tokenChar('>'); return OP_GT; }
"(๑→‿←๑)"                    { tokenChar('='); return OP_EQ; }

"‹(⁽˙ʷ˙⁾ )V( ⁽˙ʷ˙⁾ )›”       { token(and); return OP_AND; }
"(v〃∇〃)ﾊ(〃∇〃v)”           { token(or); return OP_OR; }
"⁄(⁄ ⁄•⁄-⁄•⁄ ⁄)⁄"            { token(not); return OP_NOT; }

"<º))))><"                   { token(KWarray); return ARRAY; }
"ʕ•□•ʔ"                      { token(KWstring); return STRING; }
"⊂((υ⊥υ))⊃"                  { token(KWinteger); return INTEGER; }
"(=ↀωↀ=)"                    { token(KWvar); return VAR; }
"／(≡・ x ・≡)＼"             { token(KWboolean); return BOOLEAN; }
"( ´(00)`)"                  { token(KWreal); return REAL; }

"(´･ω･)o━━━"                 { token(KWbegin); return BEG; }
"━━━o(･ω･｀)"                { token(KWend); return END; }


"♥‿♥"                        { token(KWdef); return DEF; }
"(｢･ω･)｢"                    { token(KWwhile); return WHILE; }
"◉_◉"                        { token(KWdo); return DO; }
"⊙‿⊙"                        { token(KWif); return IF; }
```

```
"⚆ _ ⚆"                         { token(KWelse); return ELSE; }
"ರ_ರ"                          { token(KWthen); return THEN; }
"ಠ‿↼"                          { token(KWfor); return FOR; }
"¬_¬"                          { token(KWof); return OF; }
"ಠ_ಠ"                          { token(KWprint); return PRINT; }
"｡◕‿◕｡"                     { token(KWread); return READ; }
"(⊃‿⊂)"                        { token(KWreturn); return RETURN; }
"◔ ⌣ ◔"                     { token(KWto); return TO; }

"(┬(●●)┬ )"                     { token(KWfalse); return FALSE; }
"(T__T)"                       { token(KWtrue); return TRUE; }
```

```
// program beginning ID inconsistent with file name
printf("%d (□°д°): Please give us A+, please~ please!! \n", linenum);}

//program beginning ID inconsistent with ending ID
printf("%d ☜(`o´): Please give us A+. \n", linenum);}

//Variable redeclare
printf("%d ◢▆▅▄▃-╰(〒皿〒)╯-▃▄▅▆◣ : I've seen that Cat Face. \n", linenum);}

// Function redeclare
printf("%d .·´¯`(>▂<)´¯`·. : I've seen that Heart Eye Face. \n", linenum);}

//Parameter redeclare
printf("%d □(≖‸↼")□ :Same name for this kind of pet. \n", linenum);}

// function beginning ID inconsistent with ending ID
printf("%d ┌∩┐(◣_◢)┌∩┐: You need a funnier ID \n", linenum);}

//Declare a wrong array: Starting number is bigger or equal than ending number
printf("%d (ノಠ益ಠ)ノ彡┻━┻ : Wrong hand instruction; Hand up and turn right \n", linenum);}

// Loop variable can not be assigned
printf("%d ⁽⁽(੭□ •Д• )੭*⁾⁾ □ₒ□ₑ ₒ□‼ : Loop Cat Face %s can not be assigned \n", linenum,$1.idname);}

// Constant variable can not be assigned
printf("%d o͡╮(□ □皿□)͡╭o͡ : Symbolic Cat Face %s can not be assigned \n", linenum,$1.idname);}

// Assign statement mismatch
printf("%d (ʶ͡ ᵕ̈ŏ̴oˇ)ʶ͡=͟ ⌨ : Assign Face statement mismatch Left: %s Right: %s \n", linenum,$1.type, $3.type);}
```

```
//Print statement operand is array type
printf("%d ( ≧Д≦) : Long Eyelash Face statement operand is in a Fish type. \n", linenum);}

//Read statement operand is array type
printf("%d ( ˙ε˙ ): Dot Cheek Face statement operand is in a Fish type.\n", linenum);}

//Variable does not exist
printf("%d ⌈RIP⌉◟◟◟◟◟◟◟◟ ❀◟(ó ̯ ò, ): This Cat Face %s is dead \n", linenum,$1);}

//Reference too much! Out of range
printf("%d (✖ ╭╮ ✖) :You know too much.\n", linenum,$1);}

//Array index wrong type
printf("%d ლ(ಠ益ಠ)ლ: The index of parts in the Fish Face %s can not be %s type \n", linenum,$1.idname,$1.type);}

// Operand after - wrong type
printf("%d (#ಠQಠ#): Operand after Sub Dance Face can not be %s type \n", linenum,$2.type);}

// Operands between * wrong type
printf("%d ˚‧º·(˚ ˃̣̣̥⌓˂̣̣̥ )‧º·˚: Operand after Mulo Dance Face can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operands between / wrong type
printf(“%d o(o･`з･´o)ﾉ: Operand between And Face can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operands between mod wrong type
printf(“%d (*￣(ｴ)￣*): Operand between Mode Dance Face can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operands between + wrong type
printf(“%d ξ(｡◕ˇ◊ˇ◕｡)ξ: Operand between Add Dance Face can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operand after < wrong type
printf(“%d ／(￣ ∈∋ ￣)＼: Operand after Hands Up and Face Right can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operand after <= wrong type
printf("%d (ↀДↀ): Operand after Fork Hand Directing to Right can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operand after = wrong type
```

```
printf("%d ｏ·°(∘˃̶͈̀﹏˂̶͈́∘)°·ｏ: Operand after Equal Face can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operand after >= wrong type
printf("%d ( ̿́ ̿̀ ̯̀ ̿̀): Operand after Hand Directing to Left can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operand after > wrong type
printf("%d （　(≪●≫)　）Д（　(≪●≫)　）　: Operand after Hands Up and Face Left can not be %s/%s type \n", linenum,$1.type,$3.type);}


// Operand between <> wrong type
printf("%d ᕙ‿ᕗ: Operand between Hands Up Dancing Face can not be %s/%s type \n", linenum,$1.type,$3.type);}

// Operand after not wrong type
printf("%d ヾ(｡□﹏□)ﾉﾞ: Operand after Not Face can not be %s type \n", linenum,$2.type);}

// Function does not exist
printf(Line %d:  o(╥﹏╥)o The %s does not exist... \n", linenum,$1);}

// Parameter mismatch
printf("%d (⊙ε   ⊙ﾉ)ﾉ: Pet Face mismatch  \n", linenum);}

// lower bound can not be negative
printf("%d (ง ͠ಠ_ಠ)ง: It’s too sad \n", linenum);}

// lower bound can not be negative
printf("%d （（●´∧｀●））: It’s too sad \n", linenum);}

// lower bound can not be bigger or equal than upper bound
printf(“%d ㅅ╭ ̀•⌂•́ ╮ㅅ : You are sadder than happiness \n", linenum);}

// Variable been assigned at the outer loop
printf(“%d ୧(๑•̀□•́)૭ : The cat is found elsewhere.\n", linenum);}

// Function type void does not need return statement
printf(“%d ヾ( ･`⌓´･)ﾉﾞ : How dare you give me that face! \n", linenum);}

// Function type is different from the return type
printf("%d (ﾉ｀Д´)ﾉ : Could you give me another breed?\n", linenum);}

// Main function can not have return statement
printf(“%d ┗(｀ム´)=3 : How dare you give me that face! \n", linenum);}
```

```
// error found. Unmatched token.
fprintf( stderr, "（（（（（BOMB））））） ⌒⌒⌒⌒⌒／(x~x)＼
 %d: %s\n", linenum, buf );
fprintf( stderr, "\n" );
fprintf( stderr, "( ✖ 〒 ✖) %s\n", yytext );
```