MODULAR DEVELOPMENT

prog.c

```
...
main ()
{
   ProcA();
   ProcB();
}
void ProcA (void){...}
void ProcB (void){...}
```

main.c

```
...
#include "mod1.h"
#include "mod2.h"
main()
{
  ProcA();
  ProcB();
}
```

DECOMPOSITION:

"TURN GROUPS OF FUNCTIONS INTO MODULES/LIBRARIES"

• mod1.h

```
...
void ProcA (void);
...
```

• mod2.h

```
...
void ProcB (void);
...
```

• mod1.c

```
...
#include "mod1.h"
void ProcA (void)
{...
}
```

• mod2.c

```
...
#include "mod2.h"
void ProcB (void)
{...
}
```

MODULES

- Unifying theme (math, graphics, input-output, ...)
- Should NOT depend on each other.
- Modifications should only expand functionality.
- COMPILATION: Compile main & all modules separately.
- LINKING: Link the object (.o) files → EXECUTABLE prgm.

⟼

- EX: "Pig Latin"

- Goal: Translate Line of text into Pig Latin.
- Rules:

1st char is consonant:

Scram ⟼ amscray

- Find first vowel.
- Add "scr" at end (and delete at beginning).
- Add "ay".

scr am ⟼ am scr ay

1st char is vowel:

@pple ⟼ appleway

- Add "way" at end.

This is Pig Latin.

⟼ isThay isway igPay atinLay.

• Parsing Process:

This_is_Pig_Latin

• Output:

isThay
isThay_isway
isThay_isway_igPay
...

➡ Pseudo-code: Describe needed functionality for program in colloquial English.

➡ TOP-DOWN program development / REFINEMENT:

```
main()
...
{ (1) Read text line.
  (2) Translate & display result.
}
...
```

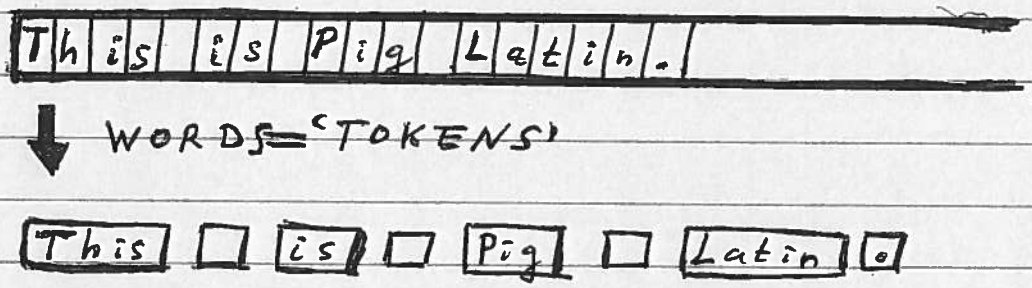
```
(1) string ReadLine(void)
{ /* READ TEXT LINE */
  string Line;
  printf("Enter text:");
  Line = GetLine();
  return(line);
}
```

② void TranslateLine (string line)

PSEUDOCODE

```
{ /* Translate into Pig Latin */
  while (there are words left)
  {
    • determine next word;
    • translate word;
  } • display translated word;
  • print '\n' character;
}
```

➡ Scanner / Parser Strategy:



• HERE: 'TOKEN' =
- group of characters \{e.g.}
- group of blank spaces \{punct. symb.}
- individual punct. symbol

➡ Scanner will extract individual tokens from text line.

- improved (2):

```
void TranslateLine (string line)
{ while (there are tokens)
  { • get next token;
    • if (token is 'English' word)
      { • replace token by Pig Latin translation;
      }
    • print token;      /* its translated version */
  }
  • print '\n' character;
}
```

→ **PROBLEM:** • Need "GetNextToken" function
(– Input: text line
– Output: next token)
• Function must know the **STATE** of scanning process – "Where is my position/character indicator?"
• NEED FOR A **GLOBAL** SCANNING **STATE VAR.**

... is_Pig_Latin...
↑ LAST TOKEN RETURNED

→ Further improved ②:

```
void TranslateLine(string Line)
{ string token;
  InitScanner(Line);
  while (!AtEndOfLine())
  { token = GetNextToken(); /*Scanner must remember STATE*/
    if (token is 'allowed' word)
    { • replace token by translation;
    }
    • print token;
  }
  • print '\n' character;
}
```

C code:

```
{ if (③ IsAllowedWord(token))
  { token = ④TranslateWord(token);
  }
  printf("%s", token);
```

③
```
bool IsAllowedWord(string token)
{ int i;
  for (i=0; i<StringLength(token); i++)
  { if (!isalnum(IthChar(token, i)))
    { return(FALSE);
    }
  }
  return(TRUE);
}
```

④ string TranslateWord (string word)
{...
• find position of 1st vowel;
if (vowel is at beginning)
{ return (concatenation (word, "way"));
}
else
{ • extract sub-string up to 1st vowel ("head");
• extract sub-string from 1st vowel to end ("tail");
• Concatenation ("tail", "head", "ay");
• return concatenation result;
}
}

• • •

GLOBAL STATE OF SCANNER:

This_is_Pig_...
↑ CURRENT LOCATION OF SCANNER

! → MUST KNOW STATE BETWEEN FUNCTION CALLS!

DEF: A VARIABLE defined outside the definitions of all functions is a **GLOBAL VARIABLE.**

- Value of a global var. can be changed by all functions.
! - Minimize the use of global variables!

Here: Use global var. to store scanner's position.

SCANNER INTERFACE

```
void InitScanner(string line);
string GetNextToken(void);
                        /* MUST KNOW SCANNER POSITION */
bool AtEndOfLine(void);
```

→ The scanner module/library provides scanning functionality & also updates the globally known state/location of the scanner; the "scopes" of all global variables are limited to the scanner module.

- Statically or privately declared variables:

```
static int spos;    /* Scanner position known only */
                    /* in module where it's defined */

static string buffer;

static int buflen;
```

{NOTE: • FUNCTIONS CAN & SHOULD BE DECLARED STATICALLY - except the main() function:

```
static return_type fctname(argument_list);
```

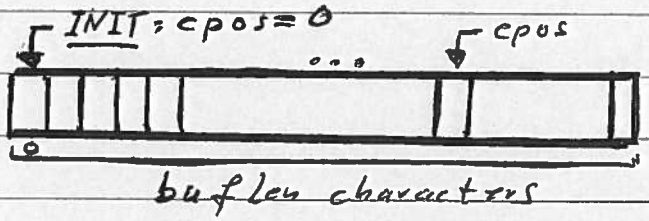
}

W6

THE SCANNER.c PROGRAM

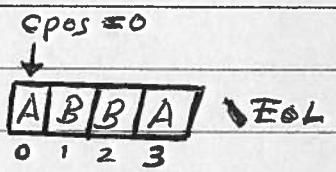
```
#include  stdio.h
          ctype.h
          genlib.h
          strlib.h
          scanner.h
```

```
/* GLOBAL, PRIVATE VARIABLES */
static string buffer;
static int buflen;
static int cpos;
```



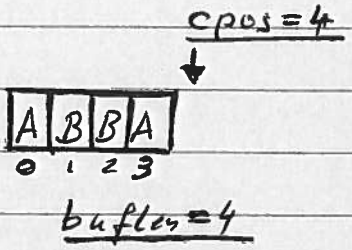
(i)

```
void InitScanner(string line)
{ buffer = line;
  buflen = StringLength(buffer);
  cpos = 0;
}
```



(ii)

```
bool AtEndOfLine(void)
{
  return (cpos >= buflen);
}
```



(iii)

```
string GetNextToken (void)
{
  char ch;
  int start;

  if (cpos >= buflen)
     Error ("No more tokens");
  ch = IthChar (buffer, cpos);
  if ( isalnum (ch))
  {  start = cpos;
     while (cpos < buflen && isalnum (IthChar(buffer, cpos)))
     {
       cpos++;
     }
     return (SubString (buffer, start, cpos-1));
  }
  else
  {  cpos++;
     return ( CharToString (ch));  /* return individual char */
  }                                /* that is not a digit, nota letter*/
}
```

"CORE PRINCIPLE"