Programming Language Processor Assignment 3

Answer the following questions and submit your report (Word or PDF) to tetsuya@shibaura-it.ac.jp before Jan. 19, 2015. The subject of your mail should be of the form "PLP Assignment 3".

Question 1

To introduce the following do-while statement to PL/0', answer the following questions.

Production rule *statement* → **do** *statement* **while** *condition*

Action A statement '**do** *statement* **while** *condition*' works as follows

1. Execute *statement*.
2. If the value of *condition* is true, go to the step 1. Otherwise, exit this loop.

Question 1-1

To add a token `do` to a set of starting tokens of *statement*, modify a function `isStBeginKey` in `compile.c` and explain the modification in your report.

Question 1-2

Modify a function `statement` in `compile.c` so that your PL/0' compiler can output object codes of Fig.1 for do-while statements. Explain the modificaiton in your report.

```
label1:  Object codes of statement
         Object codes of condition
         jpc label2
         jmp label1
label2:
```

Figure 1: Object codes for a do-while statement

Question 1-3

What does your PL/0' compiler outputs when your PL/0' compiler compiles and executes a PL/0' program do.pl0 of Fig. 2?

```
var x;
begin
   x := 0;
   do begin
      write x;
      writeln;
      x := x + 1
   end
   while x < 3
end.
```

Figure 2: A test program `do.pl0`

Question 2

Answer the following questions to add the following repeat-until statement to PL/0'.

Production rule *statement* → **repeat** *statement* **until** *condition*

Action A statement '**repeat** *statement* **until** *condition*' works as follows.

1. Execute *statement*.
2. If the value of *condition* is false, go to the step 1. Otherwise, exit this loop.

Question 2-1

Write object codes for the repeat-until statement like object codes for the do-while statement of Fig.1.

Question 2-2

Modify `getSource.h` and `getSource.c` to register two tokens `repeat` and `until` to your PL/0' compiler. Explain the modification in your report.

Question 2-3

To add a token `repeat` to a set of starting tokens of *statement*, modify a function `isStBeginKey` in `compile.c` and explain the modification in you report.

Question 2-4

Modify a function `statement` in `compile.c` so that your PL/0' compiler can output object codes for repeat-until statements. Explain the modificaiton in your report.

Question 2-5

What does your PL/0' compiler outputs when your PL/0' compiler compiles and executes a PL/0' program `repeat.pl0` of Fig.3?

```
var x;
begin
   x := 0;
   repeat begin
      write x;
      writeln;
      x := x + 1
   end
   until x=3
end.
```

Figure 3: A test program `repeat.pl0`

Question 3

Answer the following questions to add the following if-then-else statement to PL/0'.

Production rule *statement* → **if** *condition* **then** *statement*$_1$ (**else** *statement*$_2$ | ϵ)

Action A statement '**if** *condition* **then** *statement*$_1$ (**else** *statement*$_2$ | ϵ)' works as follows.

1. Evaluate *condition*.
2. If the value of *condition* is true, execute *statement*$_1$.
3. If the value of *condition* is false and *statement*$_2$ exists, execute *statement*$_2$.

Description To resolve ambiguity of the grammar of PL/0', we use the following rule.

- When we find an **else**, we relate the **else** to the nearest **then** which has not be related to any **else** yet.

Question 3-1

Write object codes for a statement '**if** *condition* **then** *statement*$_1$ **else** *statement*$_2$' like object codes for a do-while statement of Fig.1.

Question 3-2

Modify `getSource.h` and `getSource.c` to register a token `else` to your PL/0' compiler. Explain the modification in your report.

Question 3-3

Modify a function `statement` in `compile.c` so that your PL/0' compiler can output object codes for if-then-else statements. Explain the modificaiton in your report.

Question 3-4

What does your PL/0' compiler outputs when your PL/0' compiler compiles and executes a PL/0' program else.pl0 of Fig.4?

```
var x;
begin
   x := 0;
   while x<3 do begin
      if x < 1 then write 0
      else if x < 2 then write 1
      else write 2;
      writeln;
      x := x+1;
   end;
end.
```

Figure 4: A test program else.pl0

Question 4

Answer the following questions to introduce one-dimensional array to PL/0'.

Question 4-1

Explain how to modify the grammar of PL/0' to introduce one-dimensional array to PL/0'.

Question 4-2

Do you need new instructions to the PL/0' virtual machine for one-dimensional array? If you need new instructions, define thier mnemonics and their actions.

Question 4-3

Modify your PL/0' compiler so that it can support one-dimensional array. Explain the modification in your report.

Question 4-4

Write a simple test program `array.pl0` for one-dimensional array. Explain the test program and what your PL/0' compiler outputs when it compiles and executes the test program.

Question 5

Answer the following questions to introduce procedures (functions withaout any return values) to PL/0'.

We use the following statement to call a procedure with n arguments.

call *procedure*(arg_1, arg_2, ..., arg_n)

Question 5-1

Explain how to modify the grammar of PL/0' to introduce procedures to PL/0'.

Question 5-2

Do you need new instructions to the PL/0' virtual machine for procedures? If you need new instructions, define thier mnemonics and their actions.

Question 5-3

Modify your PL/0' compiler so that it can support procedures. Explain the modification in your report.

Question 5-4

Write a simple test program `proc.pl0` for procedures. Explain the test program and what your PL/0' compiler outputs when it compiles and executes the test program.

Question 6

Introduce your own idea to your PL/0' compiler.

How to submit your report

Submit an archive file which includes the following files to tetsuya@shibaura-it.ac.jp before Jan. 19, 2015.

1. Your report file report03.pdf or report03.doc (Word)
2. All source files of your PL/0' compiler
3. Makefile
4. The following test programs
   (a) array.pl0
   (b) proc.pl0
   (c) test programs for your own idea

The subject of your mail should be of the form "PLP Assignment 3".
A name of the archive file should be of the form "your_id.tgz" like "xa14000.tgz". You can make the the archive file on Linux as follows.

1. Create a directory whose name is your ID. For example, create a directory "xa14000" as follows if your ID is "xa14000".

```
% mkdir xa14000
```

2. Copy all files into the directory.
3. Make your archive file using tar command as follows.

```
% tar zcvf  xa14000.tgz  xa14000
```

Check your archive file before submission. You can expand your archive file as follows.

```
% tar zxvf  xa14000.tgz
```