National University of Singapore

SCHOOL OF COMPUTING

CS1010E — PROGRAMMING METHODOLOGY
(Semester 1 AY2015/2016)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **FOURTEEN(14)** questions and comprises **EIGHTEEN(18)** printed pages, including this page.
2. Syntax diagrams are provided for your reference. You may detach them (pp. 15–18) carefully from the rest of the paper.
3. Answer **ALL** questions.
4. Answer Section A (Questions 1 to 10) by shading the letter corresponding to the most appropriate answer on the OCR form provided.
5. Answer Sections B and C (Questions 11 to 14) within the space provided in this booklet. You may use pen or pencil to write your answers.
6. This is a **CLOSED BOOK** assessment. The maximum mark is **40**.
7. Calculators are allowed, but not electronic dictionaries, laptops, tablets, or other computing devices.
8. Do not look at the questions until you are told to do so.
9. Please write your **Student number** below. Do not write your name.

This portion is for examiner's use only.

Question	Marks	Remarks
Q11	/5	
Q12	/5	
Q13	/10	
Q14	/10	
Total	/30	

SECTION B (2 Questions : 10 Marks)
For each of the following questions, fill in each box with an appropriate statement, expression or condition. Do not use any C library functions.

11. [**5 marks**] The square root of a non-negative real number s, denoted $\sqrt{s}$, can be approximated iteratively x_n using the following numerical approximation:

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{s}{x_n}\right)$$

As an example to find $\sqrt{5}$, use the initial guess $x_0 = 2.5$ since $5/2$ is somewhat close to $\sqrt{5}$. The approximations are shown below.

$$x_1 = \frac{1}{2}\left(x_0 + \frac{s}{x_0}\right) = 2.250000$$
$$x_2 = \frac{1}{2}\left(x_1 + \frac{s}{x_1}\right) = 2.236111$$
$$x_3 = \frac{1}{2}\left(x_2 + \frac{s}{x_2}\right) = 2.236068$$

By using $s/2$ as the initial approximation, complete the following implementation of the function `mysqrt` that takes a s as a `double` value `s` and returns $\sqrt{s}$. Assume that $s > 0$.

ANSWER:

```
double mysqrt(double s) {
   double x, t;

   x = [                          ];

   do {

      [                                                    ];
      [                                                    ];

   } while ( [                    ] );

   return [                       ];
}
```

12. [**5 marks**] A sorted-and-rotated array is one which is sorted (in increasing order) and rotated left or right a number of times with wrap-around. As an example, the sorted array

1	2	3	4	5	6	7

when rotated two times to the left (with wrap-around) gives

3	4	5	6	7	1	2

The original array when rotated three times to the right (with wrap-around) gives

5	6	7	1	2	3	4

Clearly the first example of a sorted array given above is considered a sorted-and-rotated array, since rotating 0, 7, 14, ... times will revert back to the same array.

The algorithm to search for a value v within a given sorted-and-rotated array **x** containing unique values is described below.

Step 1. If v is found at the middle of array **x**, then search terminates with the middle position returned.

Step 2. If the left-half subarray is sorted, then proceed to Step 3. Otherwise, proceed to Step 4.

Step 3. If v is within the left-half sub-array, then let **x** be this left-half subarray. Otherwise, let **x** be the right-half subarray.

Step 4. If v is within the right-half sorted sub-array, then let **x** be this right-half subarray. Otherwise, let **x** be the left-half subarray.

Step 5. Repeat Steps 1 to 4 until array **x** is empty, in which case, search terminates with -1.

Complete the `binarySearchRotated` function so as to search for a value v within a sorted-and-rotated array of unique integer values of length n. The function returns the index where v occurs within the array, or -1 if v is not in the array.

ANSWER:

```
int binarySearchRotated(int x[], int n, int v) {
   int left = 0, right = n-1, mid, index = -1;

   while ((left <= right) && (index == -1)) {
      mid = (left + right) / 2;

      if (x[mid] == v) {
         index = mid;
      } else {

         if (                                          ) {

            if (                                       ) {

                                                       ;

            } else {
               left = mid + 1;
            }
         } else {

            if (                                       ) {

                                                       ;

            } else {
               right = mid - 1;
            }
         }
      }
   }

   return index;
}
```

SECTION C (2 Questions : 20 Marks)
Write your answers in the space provided.

13. [**10 marks**] Two implementations of the `strcpy` function are given below

i.
```
/* Copies string s2 to s1 */
void strcpy(char s1[],
            char s2[]) {
   int i = 0;

   while (s2[i] != '\0') {
      s1[i] = s2[i];
      i++;
   }
   s1[i] = '\0';
   return;
}
```

ii.
```
/* Copies string s2 to s1 */
void strcpy(char s1[],
            char s2[]) {
   int i = strlen(s2);

   while (i >= 0) {
      s1[i] = s2[i];
      i--;
   }
   return;
}
```

(a) [**2 marks**] What are the outcomes of the following program fragment on the two implementations of function `strcpy`?

```
char str[40] = "ABCDEFG";
strcpy(&str[1], &str[2]);
printf("%s\n", str);
```

ANSWER:

i.

ii.

(b) [**2 marks**] What are the outcomes of the following program fragment on the two implementations of function `strcpy`?

```
char str[40] = "ABCDEFG";
strcpy(&str[2], &str[1]);
printf("%s\n", str);
```

ANSWER:

i.

ii.

(c) [**1 mark**] With respect to the above program fragments in questions 13a and 13b, describe the limitations of the `strcpy` function in ten or fewer words.

ANSWER:

(d) [**5 marks**] Re-implement the `strcpy` function such that the preceding program fragment of question 13a outputs `ACDEFG`, while that of question 13b outputs `ABBCDEFG`. The function should still perform the correct string copy for any two given valid strings. Implement the function in one of two ways:

- For a maximum of 5 marks, declare only **ONE** character variable;
- For a maximum of 3 marks, there is no restriction on the variables declared. If you declare a character array, assume that string `s2` is less than 100 characters in length.

ANSWER:

```
void strcpy(char s1[], char s2[]) {
```

14. [**10 marks**] The following illustrates three working programs (reproduced from question 9) that are exactly the same apart from their indentation. In particular, the one in the middle is left-justified, while the one on the right is properly indented.

```
#include <stdio.h>

   int x(int a);
   int y(int b);

     int x(int a) {
   if (a == 1 || a == -1) {
    return a;
   } else {
   if (a < 0) {
  return y(a+1) - a;
      } else {
       return y(a-1) - a;
     }
  }
  }

  int y(int b) {
 if (b == 1 || b == -1) {
return b;
        } else {
    return x(b) + b;
          }
  }

      int main(void) {
  int val;
 scanf("%d", &val);
  printf("%d\n", x(val));
return 0;
     }
```

```
#include <stdio.h>

int x(int a);
int y(int b);

int x(int a) {
if (a == 1 || a == -1) {
return a;
} else {
if (a < 0) {
return y(a+1) - a;
} else {
return y(a-1) - a;
}
}
}

int y(int b) {
if (b == 1 || b == -1) {
return b;
} else {
return x(b) + b;
}
}

int main(void) {
int val;
scanf("%d", &val);
printf("%d\n", x(val));
return 0;
}
```

```
#include <stdio.h>

int x(int a);
int y(int b);

int x(int a) {
   if (a == 1 || a == -1) {
      return a;
   } else {
      if (a < 0) {
         return y(a+1) - a;
      } else {
         return y(a-1) - a;
      }
   }
}

int y(int b) {
   if (b == 1 || b == -1) {
      return b;
   } else {
      return x(b) + b;
   }
}

int main(void) {
   int val;
   scanf("%d", &val);
   printf("%d\n", x(val));
   return 0;
}
```

In this question, you are to implement two functions `readAndTrim` and `autoIndent` so as to output a properly indented program using a code fragment as follows:

```
char s[MAX];
readAndTrim(s);
autoIndent(s);
printf("%s", s);
```

You may make the following assumptions for the rest of this question.

- The `MAX` constant has been defined;
- Library headers `<stdio.h>`, `<string.h>` and `<boolean.h>` have been included;
- The program is run using the command `.\a.out < someprog.c` where `someprog.c` is a valid working C program containing all `{` and `}` curly braces necessary to determine indentation (even for single statement blocks).

(a) [**5 marks**] Complete the function `readAndTrim` that reads the contents of a valid C program as input, trims off the leading spaces and stores the left-justified program into the argument string `s`.

A skeleton function that reads the entire file (including all whitespaces) is given. Complete the function to generate a left-justfied program in the string `s`.

ANSWER:

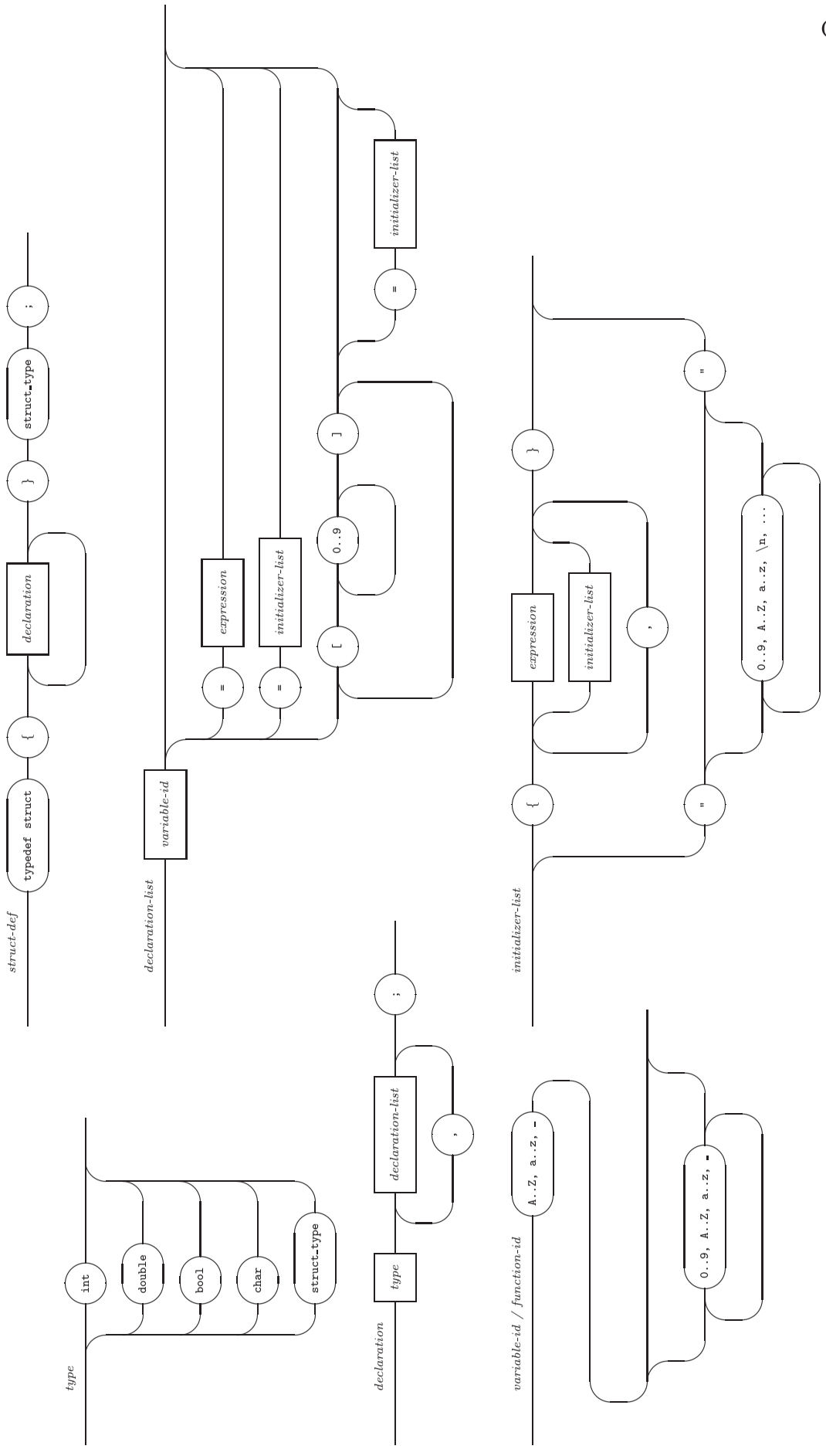
```
void readAndTrim(char s[]) {
   char c;

   /* declare other variables here */




   while (scanf("%c", &c) != EOF) {

   }


   return;
}
```

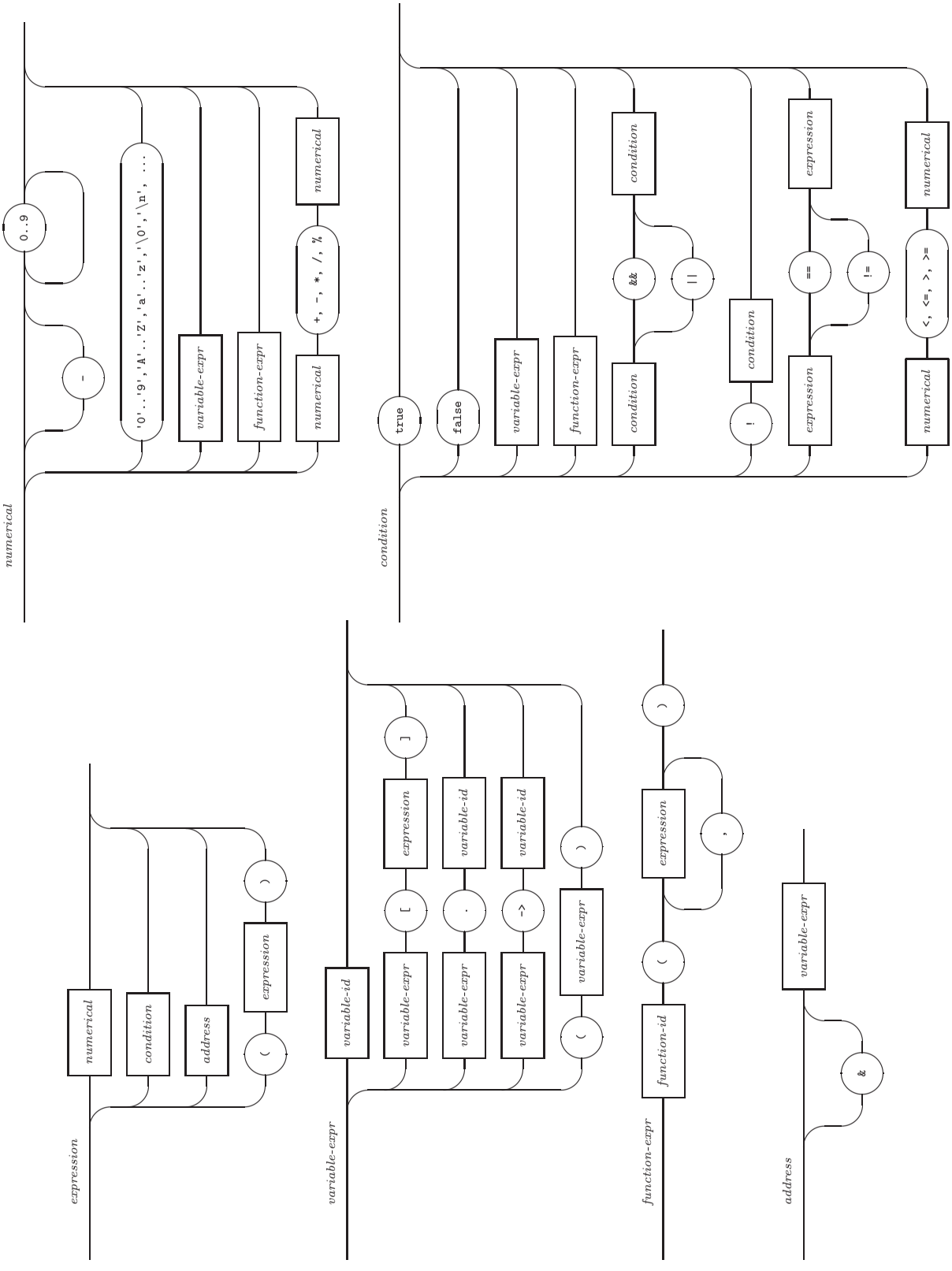
(b) **[5 marks]** Implement the function `autoIndent` that takes as argument a string `s` containing the left-justified program and performs the appropriate indentation. Take note of the following:

- Use the `{` and `}` curly braces to determine the indentation level;
- Assume that `}` occurs at the start of a line; while `{` occurs at the end of a line;
- Use an indentation spacing of three spaces; you may indicate a space using ␣.

ANSWER:

```
void autoIndent(char s[]) {
```


struct-def
typedef struct
{
declaration
}
struct_type
;
declaration-list
variable-id
=
expression
=
initializer-list
[
0..9
]
=
initializer-list
initializer-list
{
expression
initializer-list
,
}
"
0..9, A..Z, a..z, \n, ...
"
type
int
double
bool
char
struct_type
declaration
type
declaration-list
,
;
variable-id / function-id
A..Z, a..z, _
0..9, A..Z, a..z, _


numerical
0..9
-
'0'..'9','A'..'Z','a'..'z','\0','\n', ...
variable-expr
function-expr
numerical
numerical
+, -, *, /, %
condition
true
false
variable-expr
function-expr
condition
&&
||
condition
!
condition
expression
==
!=
expression
numerical
<, <=, >, >=
numerical
expression
numerical
condition
address
(
expression
)
variable-expr
variable-id
[
expression
]
variable-expr
.
variable-id
variable-expr
->
variable-id
variable-expr
(
)
function-expr
function-id
(
expression
,
)
address
variable-expr
&

statement
assignment-stat
control-stat
function-stat
return-stat

assignment-stat
variable-expr
=
expression
;
variable-expr++
variable-expr--

function-stat
function-expr
;

return-stat
return
;
expression

control-stat
if-else-stat
while-stat
do-while-stat
for-stat

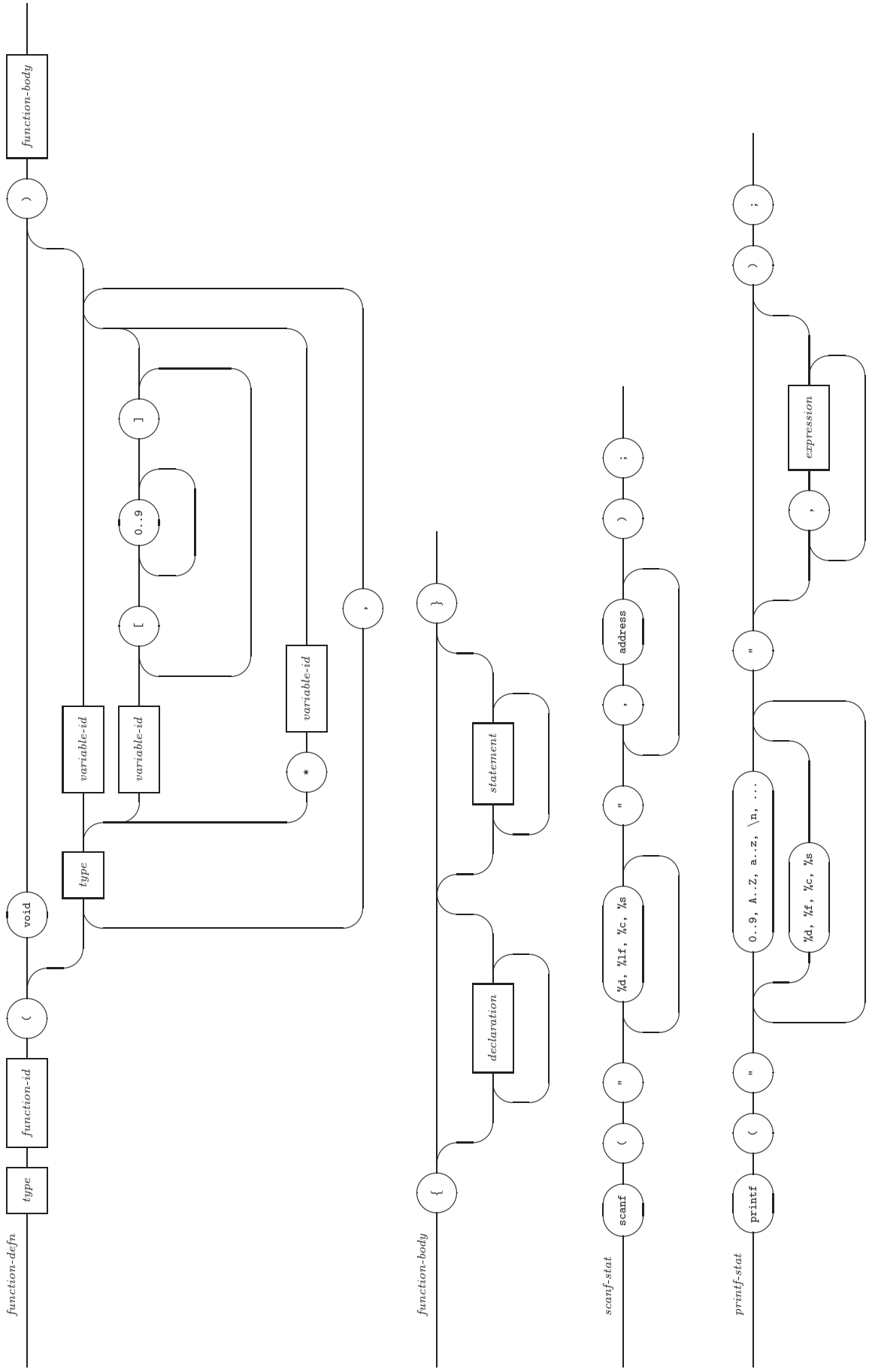
stat-block
statement
{
statement
}

if-else-stat
if
(
condition
)
stat-block
else
stat-block

while-stat
while
(
condition
)
stat-block

do-while-stat
do
stat-block
while
(
condition
)
;

for-stat
for
(
assignment
;
condition
;
assignment
)
stat-block


function-defn
type
function-id
(
void
type
variable-id
variable-id
[
0..9
]
variable-id
*
,
)
function-body
{
declaration
statement
}
scanf-stat
scanf
(
"
%d, %lf, %c, %s
"
,
address
)
;
printf-stat
printf
(
"
0..9, A..Z, a..z, \n, ...
%d, %f, %c, %s
"
,
expression
)
;