Term Project TA Session

[CS 420] Compiler Design

TA Kyuho Son

Debugger Implementation

- Goal
  - Mini-C language interpreter
    - Mini-C?
    - Scope : enough to handle the sample code (subset of C89)
  - Features
    - Interpretation
    - Built-in function (printf without include)
    - Debug CLI commands

Interpreter Implementation

- Interpretation
  - Typical interpreter
    - Building AST in run-time and execution
    - No trace feature
  - For term project scope
    - AST building : Your choice
    - Should have the feature of tracing values of variables
    - More like ‘debugger’

Interpreter Implementation

Example input code

```c
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

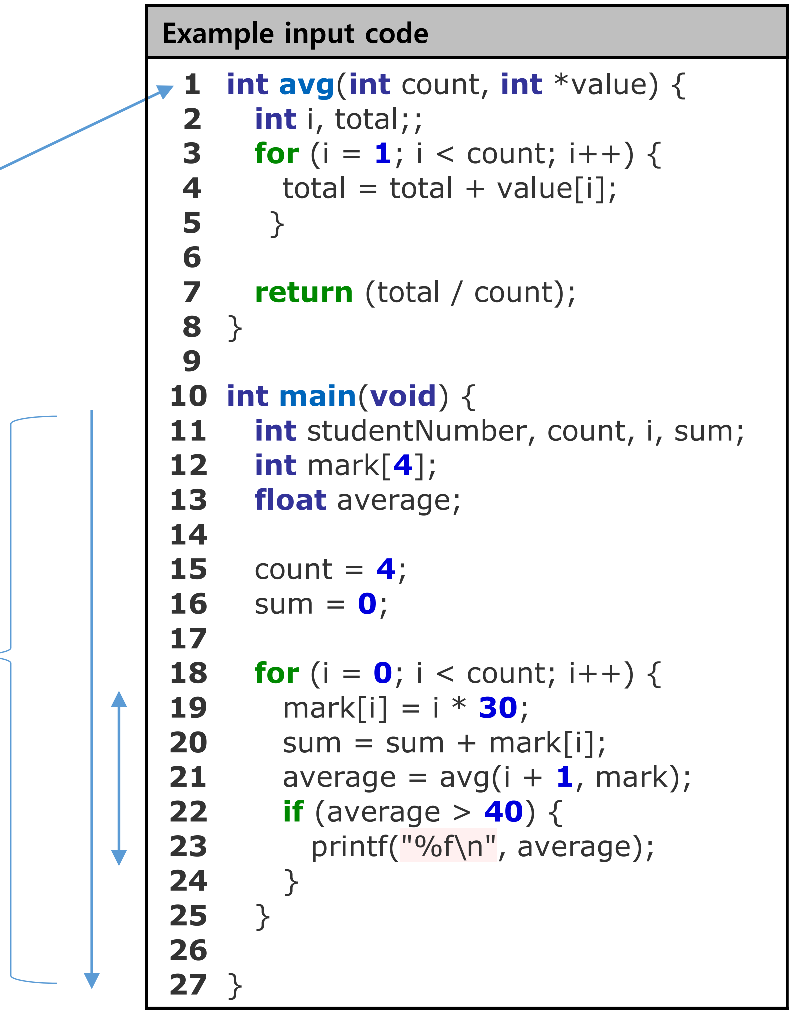
- Implementation scope
  - ➢ Variable types (int, float)
  - ➢ Variable declaration
  - ➢ Variable assignment
  - ➢ Calculation (+ , - , *, /, + +)
  - ➢ Comparison (>, <)
  - ➢ Type casting (int ↔ float)
  - ➢ Flow control (for, if)
  - ➢ Pointer
  - ➢ Function call and return
  - ➢ 1-dim array
  - ➢ printf(); function (with built-in)
  - ➢ brackets…
  - ➢ TBD : Recursive function call?

Interpreter Implementation

- CLI Commands
  - next [line number]
    - ➢ The line number of statements are executed
  - print [variable name]
    - ➢ print the value of the variable in current scope
  - trace [variable name]
    - ➢ print the history of values of the variable in current scope

Interpreter Implementation

- Terminology
  - Line number
    - ➢ Meaning ①
      : *code line*
    - ➢ Meaning ②
      : *execution lines*

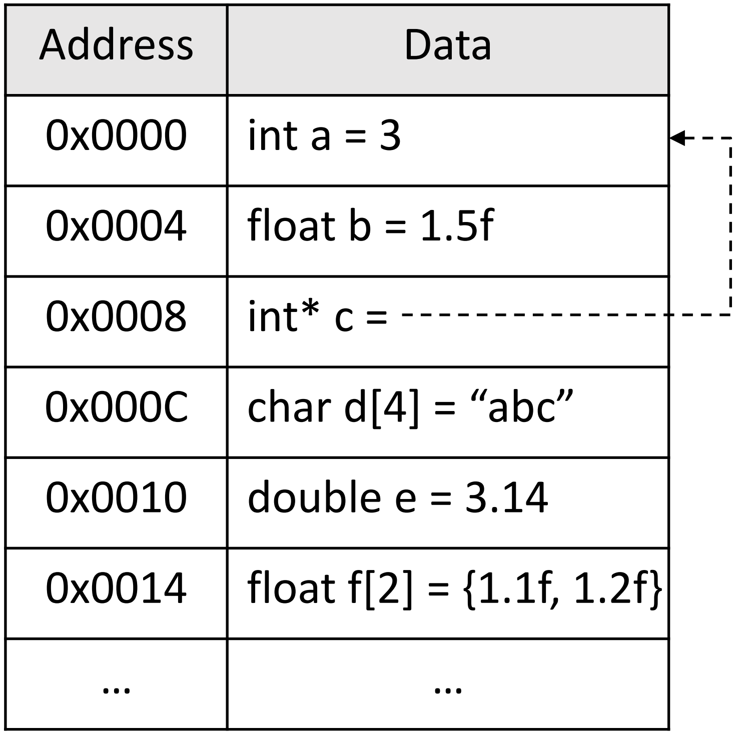


Example input code

```
1  int avg(int count, int *value) {
2    int i, total;;
3    for (i = 1; i < count; i++) {
4      total = total + value[i];
5     }
6
7    return (total / count);
8  }
9
10 int main(void) {
11   int studentNumber, count, i, sum;
12   int mark[4];
13   float average;
14
15   count = 4;
16   sum = 0;
17
18   for (i = 0; i < count; i++) {
19     mark[i] = i * 30;
20     sum = sum + mark[i];
21     average = avg(i + 1, mark);
22     if (average > 40) {
23       printf("%f\n", average);
24     }
25   }
26
27 }
```

Interpreter Implementation

- Terminology
  - Value
    - a = 3
    - b = 1.5
    - c = 0x0000
    - d = 0x000C
    - e = 3.14
    - f = 0x0014
    - *c = 3
    - d[2] = ‘c’
    - d[3] = null character
    - f[0] = 1.1

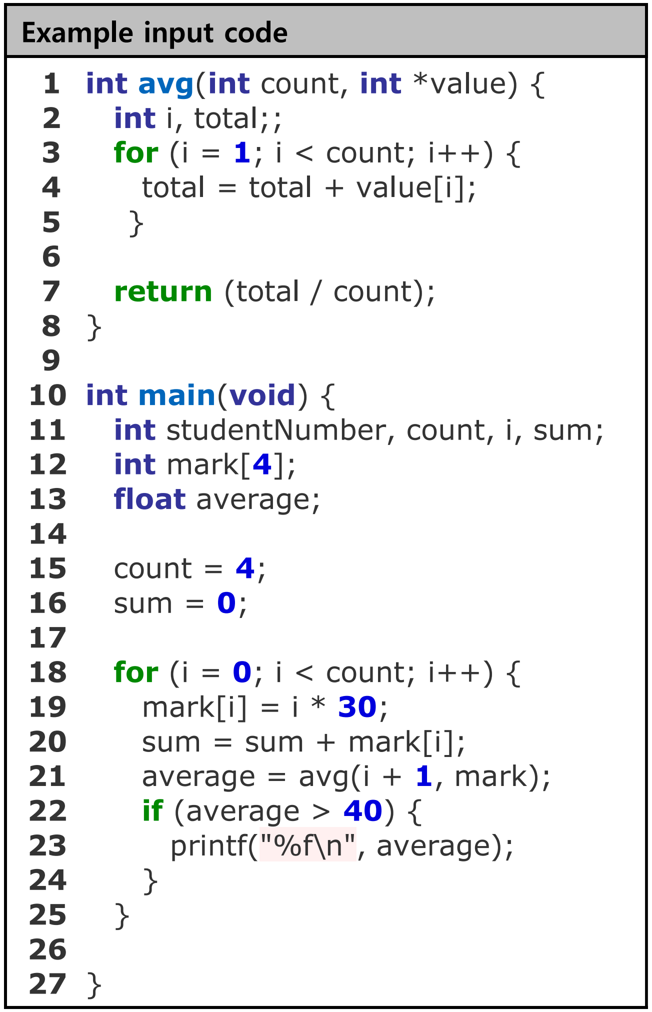


Address	Data
0x0000	int a = 3
0x0004	float b = 1.5f
0x0008	int* c =
0x000C	char d[4] = “abc”
0x0010	double e = 3.14
0x0014	float f[2] = {1.1f, 1.2f}
…	…

Interpreter Implementation

- Terminology
  - Scope
    - Visibility of the variable

      (Visible / Invisible)

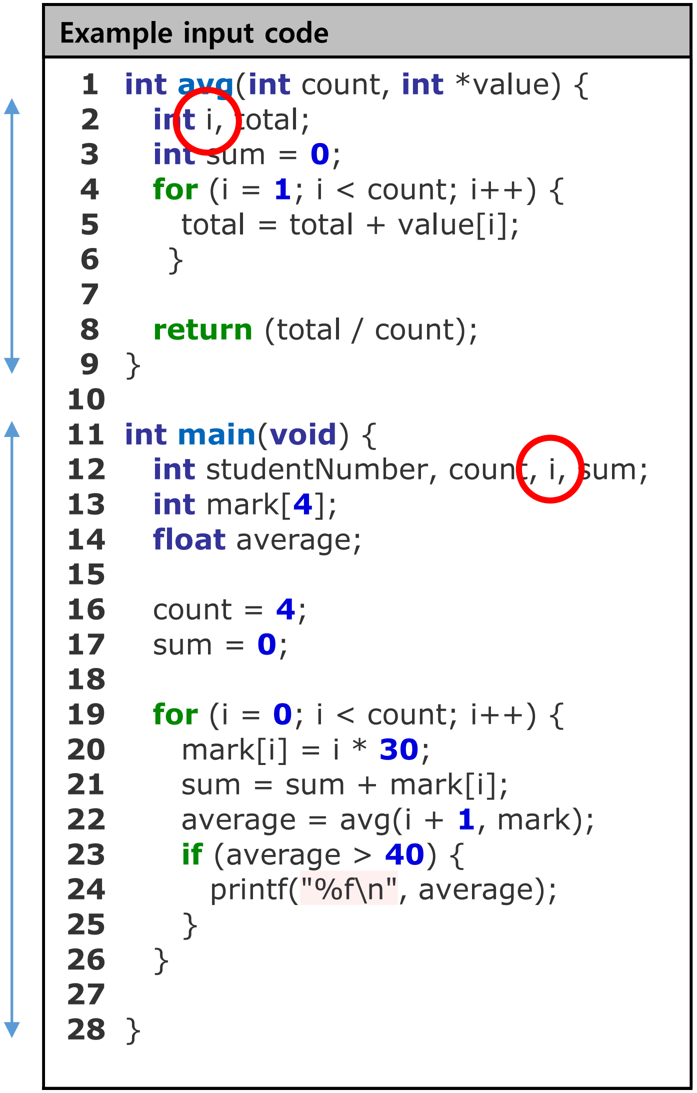


Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - Scope
    - Scope of var *i*

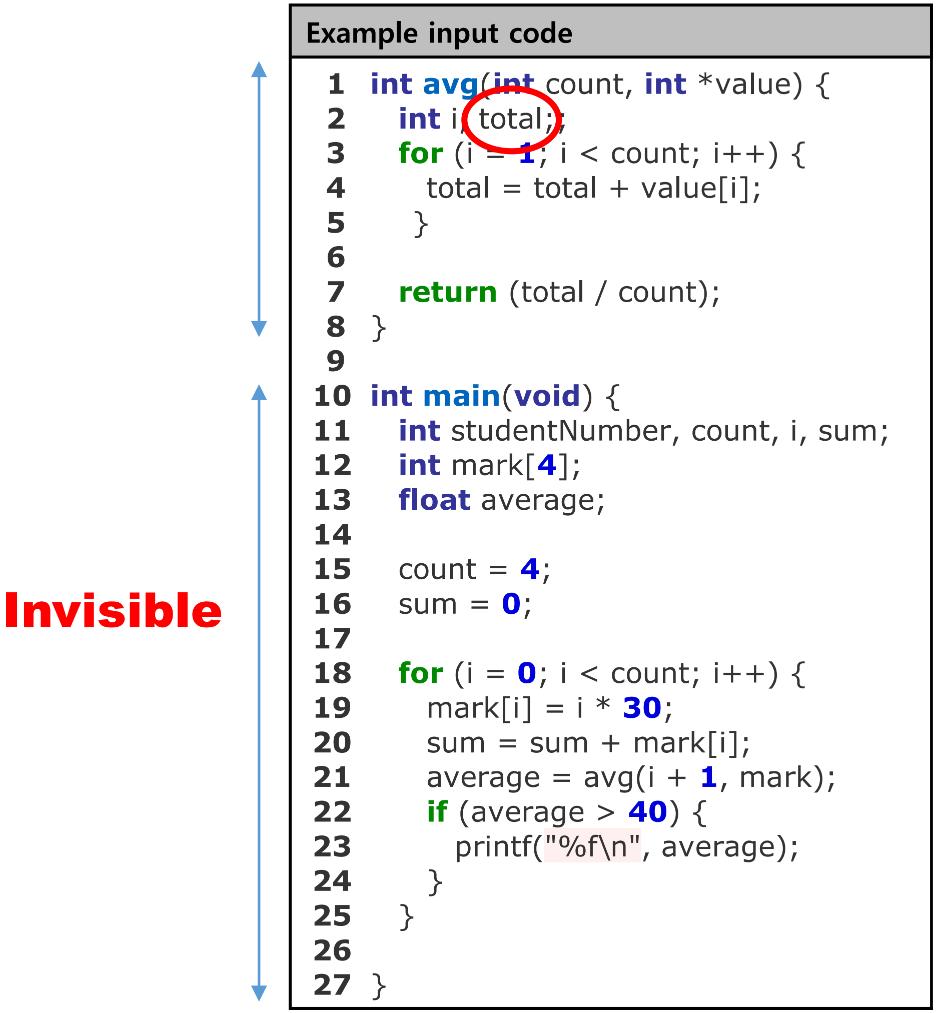


Example input code

```
int avg(int count, int *value) {
  int i, total;
  int sum = 0;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - Scope
    - Scope of var *total*

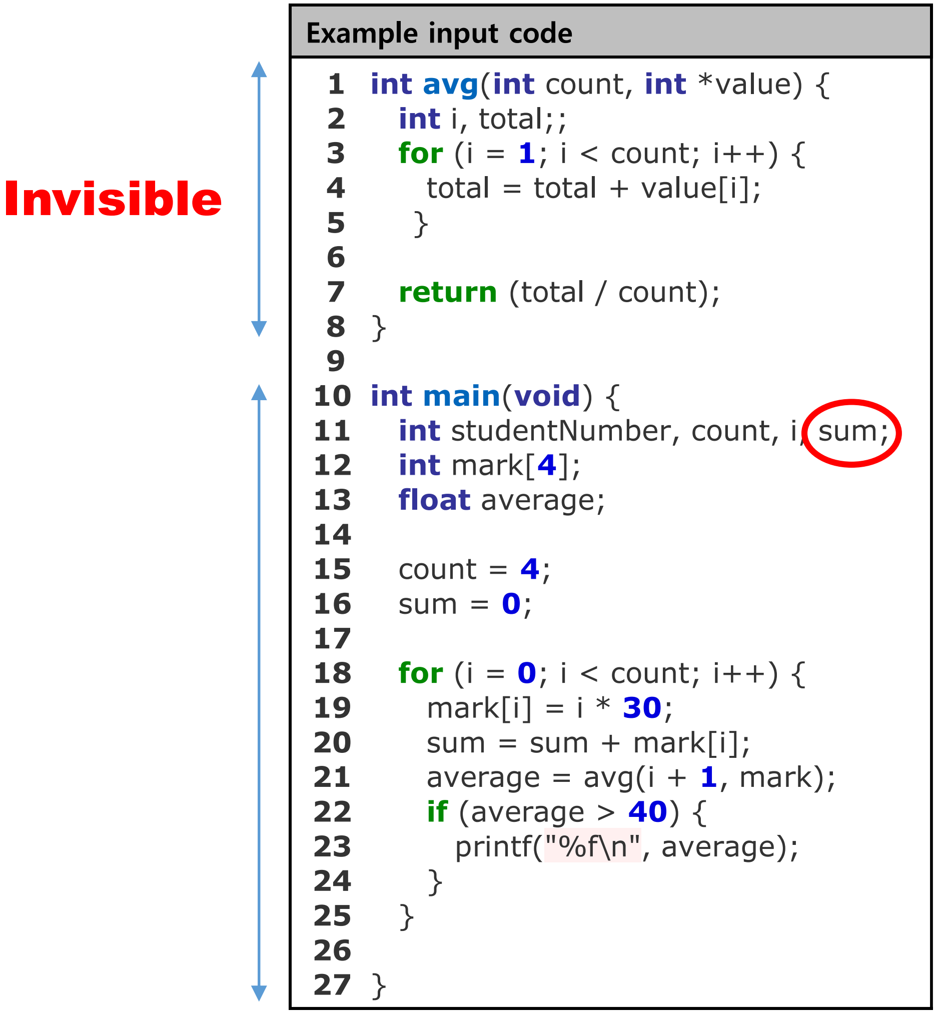


Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - Scope
    - Scope of var *sum*



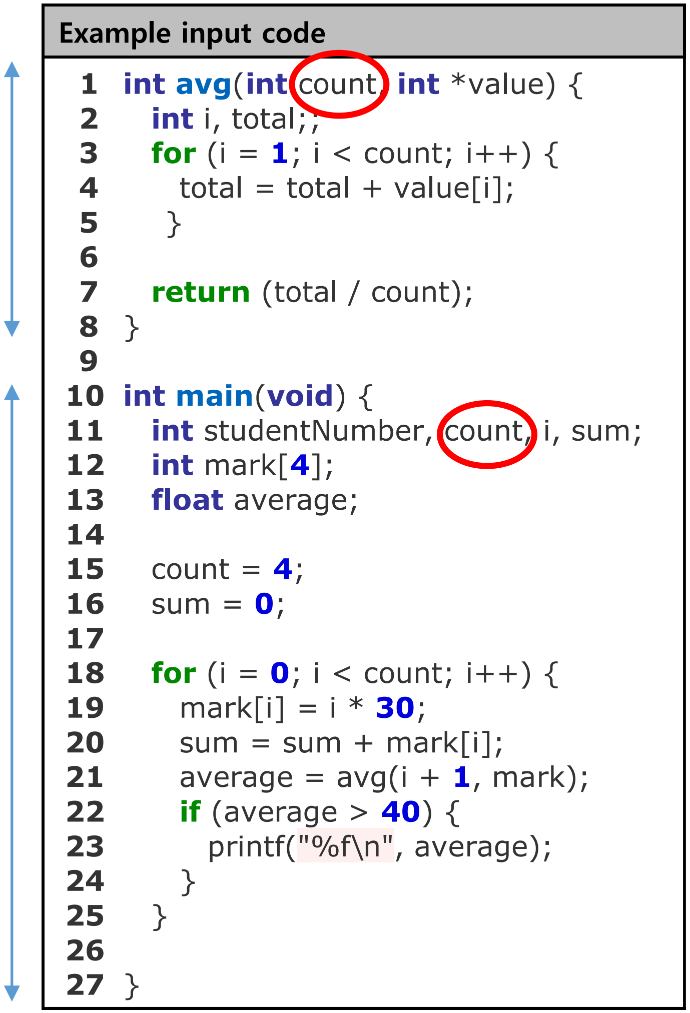
Invisible

Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - Scope
    - Scope of var *count*



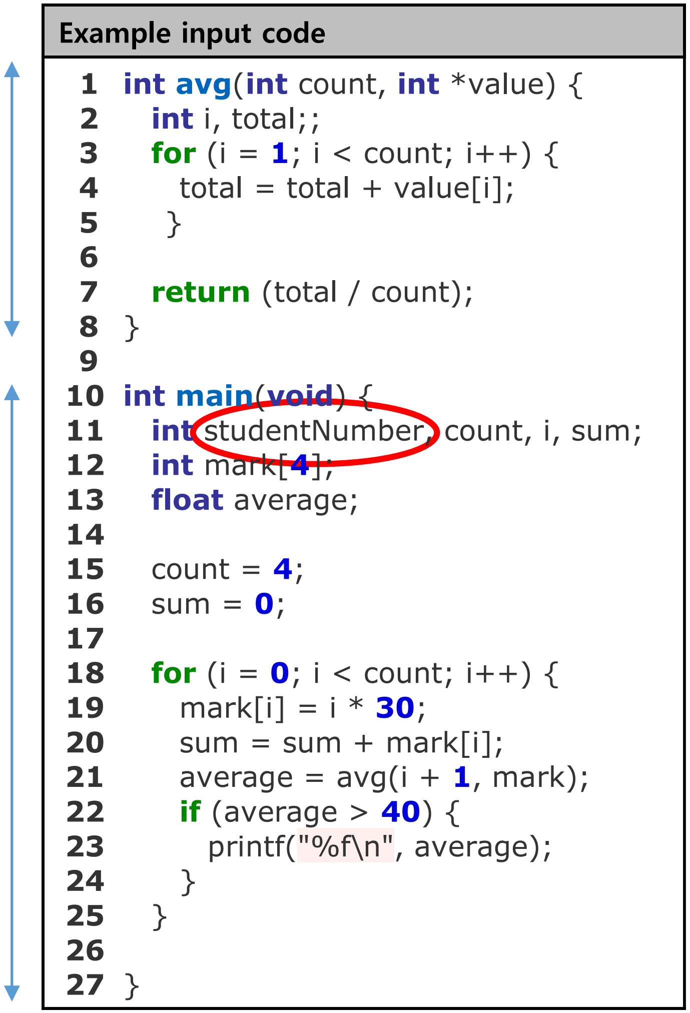
Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - Scope
    - Scope of var *studentNumber*

Invisible

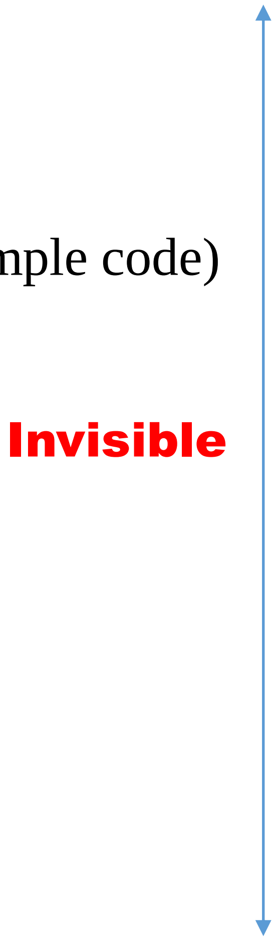


Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - Scope
    - Scope of var *stdev* (Not exist in the sample code)



Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - History
    - ➢ Life time of history (declaration ~ expiration)

      You do not need to maintain histories of expired variables!
    - ➢ Variable declaration (N/A on declaration w/o assignment)
    - ➢ Value assignment

Example input code

```
int avg(int count, int *value) {
  int i, total;;
  for (i = 1; i < count; i++) {
    total = total + value[i];
   }

  return (total / count);
}

int main(void) {
  int studentNumber, count, i, sum;
  int mark[4];
  float average;

  count = 4;
  sum = 0;

  for (i = 0; i < count; i++) {
    mark[i] = i * 30;
    sum = sum + mark[i];
    average = avg(i + 1, mark);
    if (average > 40) {
      printf("%f\n", average);
    }
  }

}
```

Interpreter Implementation

- Terminology
  - History of *i* in this line

Meaning ②

Meaning ①

Code line	Value
2	N/A
4	1
4	2
4	3
…	…

Example input code

```
 1  int avg(int count, int *value) {
 2    int i, total;;
 3    for (i = 1; i < count; i++) {
 4      total = total + value[i];
 5     }
 6
 7    return (total / count);
 8  }
 9
10  int main(void) {
11    int studentNumber, count, i, sum;
12    int mark[4];
13    float average;
14
15    count = 4;
16    sum = 0;
17
18    for (i = 0; i < count; i++) {
19      mark[i] = i * 30;
20      sum = sum + mark[i];
21      average = avg(i + 1, mark);
22      if (average > 40) {
23        printf("%f\n", average);
24      }
25    }
26
27  }
```

Interpreter Implementation

- Other features
  - Syntax error handling : stop interpretation
  - [Optional] Run-time error handling
  - [Optional] Register allocation
  - [Optional] Further features in C language

Implementation of optional features is
not a mandatory but an option!

Interpreter Implementation

- Team building
    - 3 members in a team
    - Gather team members you want to do the project together and a representative send TA a mail that contains student IDs and names of all team members (ableman@kaist.ac.kr)
    - You can use team building board in KLMS
    - Due : 6th Nov. Tue.
    - For those who couldn’t assemble a team till the due, TA will assign their team
    - For all the products from the team work, contribution of members should be specified

Interpreter Implementation

- TA will do the best effort reviewing your products
  - TA reviews all source code quite carefully
    (Actually it is necessary to give partial scores for all products)
  - Late submission will always be better than nothing
  - If your source code does not operate, you will lose most, but still much better than nothing

So, do your best!!

QnA