TERM PROJECT TA SESSION

[CS 420] Compiler Design

TA Kyuho Son

TA Wonyoung Lee

Interpreter Implementation

- Goal
  - Semi-C language interpreter
    - Semi-C?
    - Scope : equivalent to the sample code
  - Features
    - Interpretation
    - Built-in function (printf)
    - 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

Interpreter Implementation

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);
    }
  }

}
```

- 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…

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    int sum = 0;
4    for (i = 1; i < count; i++) {
5      total = total + value[i];
6     }
7
8    return (total / count);
9  }
10
11 int main(void) {
12   int studentNumber, count, i, sum;
13   int mark[4];
14   float average;
15
16   count = 4;
17   sum = 0;
18
19   for (i = 0; i < count; i++) {
20     mark[i] = i * 30;
21     sum = sum + mark[i];
22     average = avg(i + 1, mark);
23     if (average > 40) {
24       printf("%f\n", average);
25     }
26   }
27
28 }
```

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

```c
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 *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*

Invisible

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 *sum*

Invisible

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 *count*

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 *studentNumber*

Invisible

Example input code

```
1  int avg(int count, int *value) {
2    int i, total;
3    int sum = 0;
4    for (i = 1; i < count; i++) {
5      total = total + value[i];
6     }
7
8    return (total / count);
9  }
10
11 int main(void) {
12   int studentNumber, count, i, sum;
13   int mark[4];
14   float average;
15
16   count = 4;
17   sum = 0;
18
19   for (i = 0; i < count; i++) {
20     mark[i] = i * 30;
21     sum = sum + mark[i];
22     average = avg(i + 1, mark);
23     if (average > 40) {
24       printf("%f\n", average);
25     }
26   }
27
28 }
```

Interpreter Implementation

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

Invisible

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
  - 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;
  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
  - History of *i* in this line

Meaning ①

Meaning ②

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    int sum = 0;
4    for (i = 1; i < count; i++) {
5      total = total + value[i];
6     }
7
8    return (total / count);
9  }
10
11 int main(void) {
12   int studentNumber, count, i, sum;
13   int mark[4];
14   float average;
15
16   count = 4;
17   sum = 0;
18
19   for (i = 0; i < count; i++) {
20     mark[i] = i * 30;
21     sum = sum + mark[i];
22     average = avg(i + 1, mark);
23     if (average > 40) {
24       printf("%f\n", average);
25     }
26   }
27
28 }
```

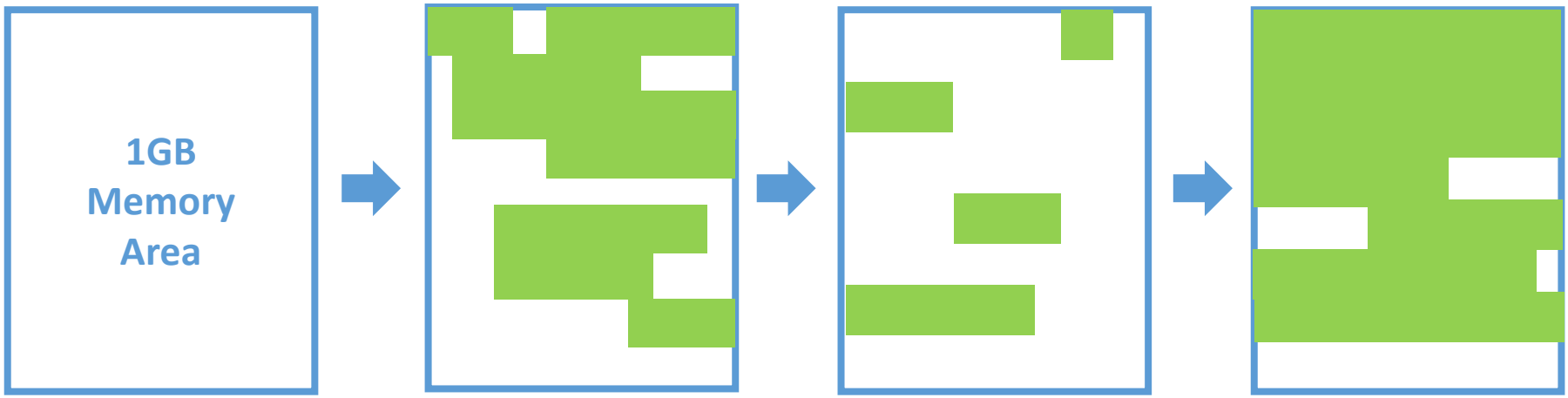
Interpreter Implementation

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

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

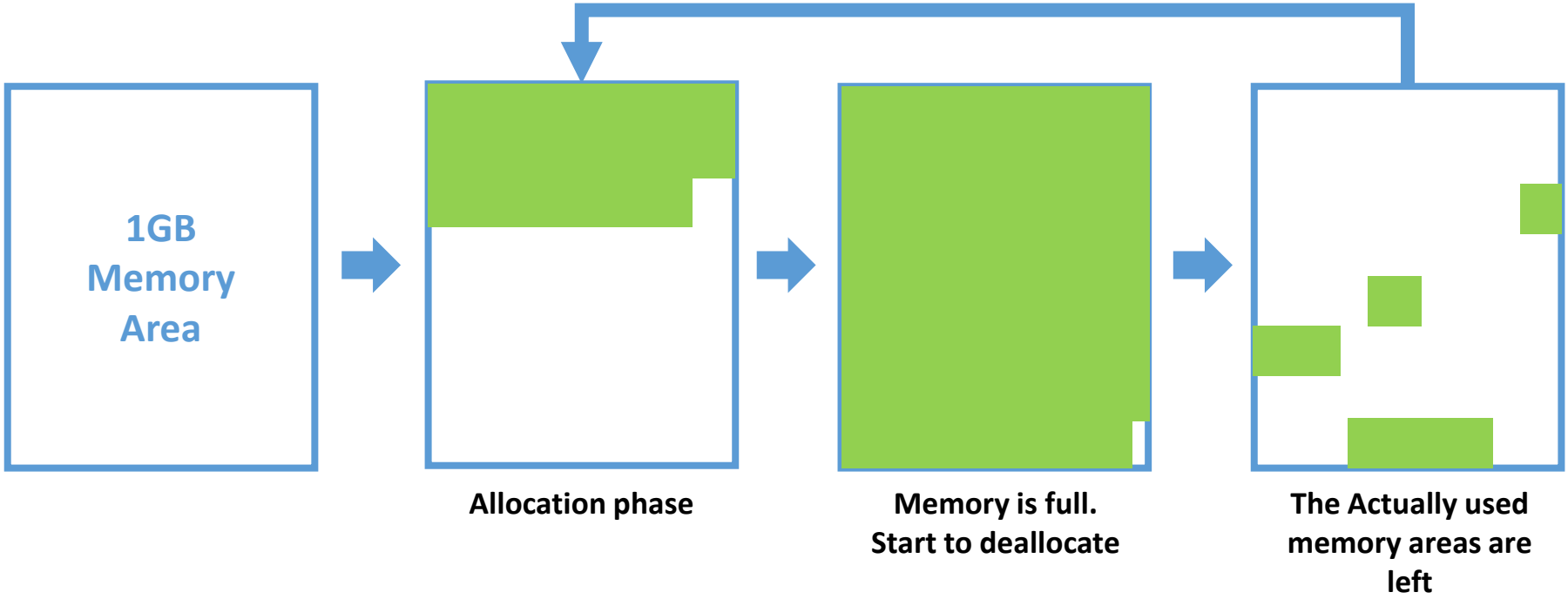
Memory management

- Schedule
  - Each schedule limits its memory usage within 1GB
  - Randomized schedule
    - Randomized sequence of calling allocation or deallocation function
    - Up to 2000 calls to exhaustively utilize 1GB memory



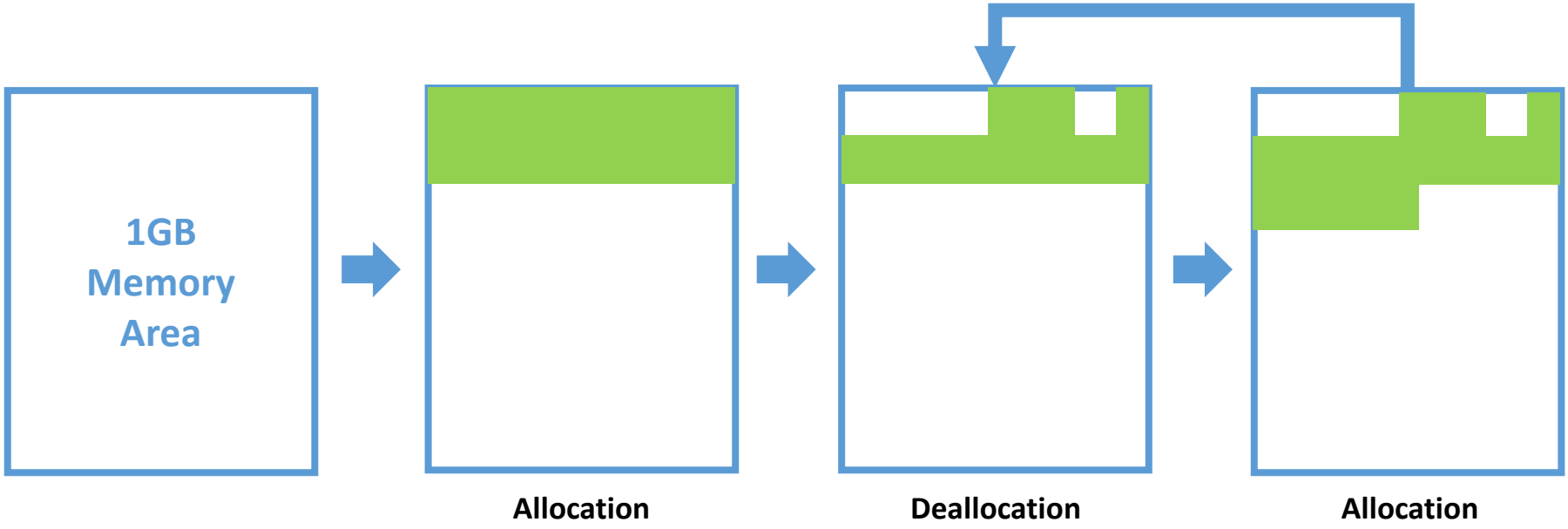
Memory management

- Schedule
  - Greedy schedule
    - No deallocation before almost fully filling 1GB memory
    - If the memory is full, deallocations occur in almost all memory area



Memory management

- Schedule
  - Back-and-forth schedule
    - ➢ Allocations and deallocations are called alternatively



Memory management

- Unit Action
  - A series of function calls that should be performed atomically
  - Each action has its own cost
    - The cost is used as a metric to measure overall performance
    - i.e. the overall performance is calculated by

$$Overall\ performace = \sum_{\boldsymbol{i}} (The\ number\ of\ calls\ of\ Action\ \boldsymbol{i}) \times (The\ cost\ of\ Action\ \boldsymbol{i})$$

QnA