CS1010 Programming Methodology

http://www.comp.nus.edu.sg/~cs1010/

WEEK 11

PE2

NUS National University of Singapore | School of Computing

Leading The World With Asia's Best

Automatic Grading (Cohort)

Ex1: Spy		
Grade A:	45	21%
Grade B:	4	1.9%
Grade C:	16	7.5%
Grade D:	22	10.3%
Grade E:	107	50%
Grade F:	20	9.3%

Ex2: Block Buster		
Grade A:	4	1.9%
Grade B:	11	5.2%
Grade C:	6	2.8%
Grade D:	19	8.9%
Grade E:	123	57.8%
Grade F:	50	23.5%

Spy: Task Statement (1/12)

- Follow the given encryption process to encrypt a given message based on a given keyword.
  - Step 1: Keyword Simplification
    - Remove duplicated letters
    - Example: mississippi → misp
  - Step 2: Alphabet Creation
    - Create a new alphabet with the simplified keyword + the rest of the letters in the original alphabet in alphabetical order.
    - Example: **misp**abcdefghjlknoqrtuvwxyz
  - Step 3: Encryption
    - Replace the letters in the message based on the new alphabet: The i-th letter in original alphabet → The i-th letter in new alphabet
    - Example: SuperSpy → RunaqRny

Practice exercises with similar context:
S08P07/08: Info Sec and Cryptography I/II

Spy: Task Statement (2/12)

- Inputs:
  - keyword: a string consisting of lowercase English letters
  - message: a string consisting of lowercase and/or uppercase English letters
- Output:
  - result: a string representing the encrypted message
- Intermediate results
  - keyword: a string representing the simplified keyword
  - alphabet: a string representing the new alphabet

Spy: Task Statement (3/12)

- The main function is given.

- You are to implement the following functions:
  - Step 1: **void simplify(char keyword[])**
    - Simplify keyword and store the result back in keyword
  - Step 2: **void createAlphabet(char keyword[], char alphabet[])**
    - Create an alphabet based on keyword
  - Step 3: **void encrypt(char message[], char alphabet[], char result[])**
    - Encrypt message based on alphabet, and store the encrypted message in result

- Hint 1: You may skip step 1 at first.

Spy: Step 2: Alphabet Creation (4/12)

- **void createAlphabet(char keyword[], char alphabet[])**
  - Create a new alphabet with the simplified keyword + the rest of the letters in the original alphabet in alphabetical order.
  - Hint 2: There is no need to store the original alphabet explicitly.

- Algorithm
  - Copy the keyword into alphabet
    - Copy letter by letter or use strcpy()
  - Fill in the rest of the letters
    - For each letter from 'a' to 'z', check if it has appeared in keyword by comparing it with the letters in the keyword or use strchr()
    - If the letter has not appeared, add the letter to the alphabet

Spy: Step 2: Alphabet Creation (5/12)

- How to go through the letters in alphabetical order?

```
for (ch = 'p'; ch <= 't'; ch++)
  printf("ch = %c\n", ch);
```

Demo #1 from Unit #16

- How to copy letters from one array to another?

Demo #5 from Unit #16

```
for (i = 0; i < len; i++) {
   switch (toupper(str[i])) {
      case 'A': case 'E':
      case 'I': case 'O': case 'U': break;
      default: newstr[count++] = str[i];
   }
}

newstr[count] = '\0';
```

Spy: Step 2: Alphabet Creation (6/12)

```
void createAlphabet(char keyword[], char alphabet[]){
  int i = strlen(keyword);
  char ch = 'a';

  strcpy(alphabet, keyword);

  while (i < ALPHABET_SIZE) {
    if (strchr(keyword, ch) == NULL)
      alphabet[i++] = ch;
    ch++;
  }

  alphabet[ALPHABET_SIZE] = '\0';
}
```

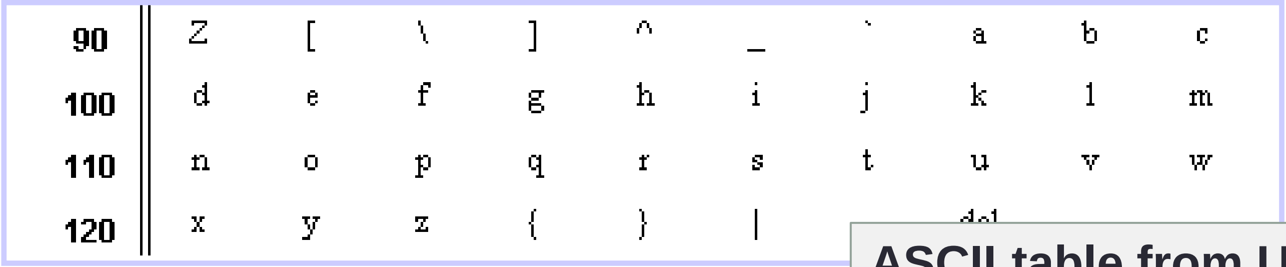
Spy: Step 3: Encryption (7/12)

- **void encrypt(char message[], char alphabet[], char result[])**
  - Replace the letters in the message based on the new alphabet: The i-th letter in original alphabet → The i-th letter in new alphabet
  - Hint 2: There is no need to store the original alphabet explicitly.

- Algorithm
  - Encrypt the message letter by letter
    - For each letter in the message, find out its position in the original alphabet
    - Check the new alphabet for the letter at the same position
    - Copy the letter in the new alphabet into result (without changing the case of the letter)

Spy: Step 3: Encryption (8/12)

- How to find the position of a letter?



90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}	|		del		

ASCII table from Unit #16

```
sum += ch – '0'
```

Ex1 from Week 9 Activities

- How to obtain the letter at the same position of the new alphabet?
  - If a letter is at position i in the original alphabet, the new letter is at alphabet[i]!

Spy: Step 3: Encryption (9/12)

```
void encrypt(char message[], char alphabet[],
             char result[]){
  int i;

  for (i = 0; i < strlen(message); i++){
    if (isupper(message[i]))
        result[i] = toupper(alphabet[message[i]-'A']);
    else
        result[i] = alphabet[message[i]-'a'];
  }

  result[i] = '\0';
}
```

Spy: Step 1: Keyword Simplification (10/12)

- **void simplify(char keyword[])**
  - Remove duplicated letters

- Algorithm
  - Copy a letter (back to the same array) if it has not appeared before
    - For letter i in the keyword, check all the letters before it
    - If it has not appeared, copy the letter back to the same array (after other letters that have been copied).

Spy: Step 1: Keyword Simplification (11/12)

- How to check whether a letter has appeared before?
  - Part of S05P02: Set Containment
  - A set X is a subset of another set Y if all the elements in X has appeared in Y. Therefore, we need to check whether a particular element in X has appeared in Y.
  - In this step, the particular element in X is the letter we are looking at, and Y is all the letters before that letter.
- How to copy letters back the same array?
  - Similar to Step 2 except that the two arrays are the same.

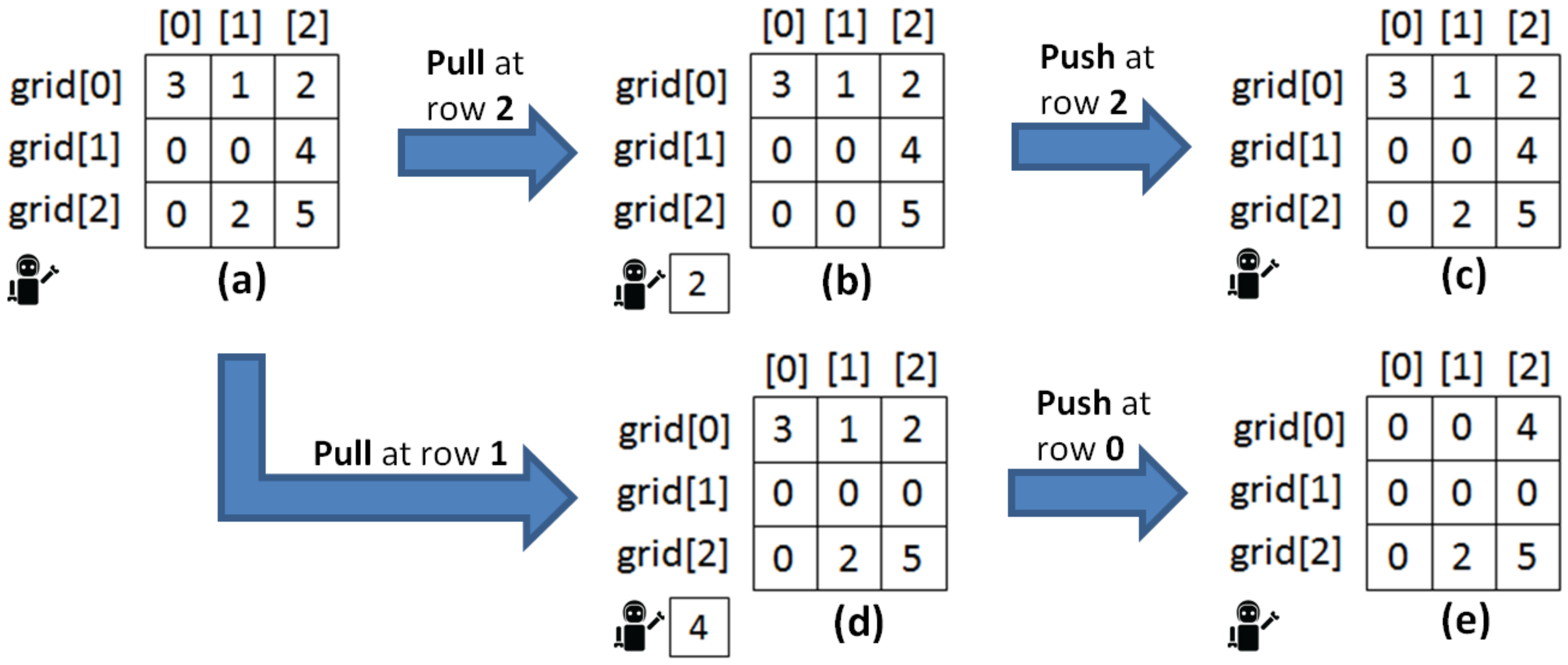
Spy: Step 1: Keyword Simplification (12/12)

```
void simplify(char keyword[]){
  int i, j, count = 0;

  for (i = 0; i < strlen(keyword); i++){
    for (j = 0; j < i; j++)
      if (keyword[i] == keyword[j])
        break;

    if (j == i)
      keyword[count++] = keyword[i];
  }

  keyword[count] = '\0';
}
```

Block Buster: Task Statement (1/12)

- Given a grid of blocks and a sequence of row indexes for actions, perform the actions on the grid.
- Two possible actions: Pull and Push
  - Pull in row i: pull the leftmost block from row i and hold it.
    - This action fails if the row is empty.
  - Push in row i: push the block (retrieved from the previous Pull action) into row i
    - The block destroys all blocks of a smaller weight until it reaches a block it cannot destroy or the edge of the grid.
    - This action fails if the row is full and it cannot even destroy the leftmost block.
- The sequence of actions is always in an alternating manner starting from a Pull

Block Buster: Task Statement (2/12)



Block Buster: Task Statement (3/12)

- Inputs:
  - The size of the grid and the values in the grid
  - The number of row indexes for the actions and the row indexes
- Output:
  - Whether the game can be completed
  - The score (i.e., total weight of all blocks destroyed)
  - The number of blocks left
  - The final state of the grid
- Intermediate results
  - The block retrieved from a Pull action
  - The partial score

Block Buster: Task Statement (4/12)

- You are to implement the following functions:
  - **readInputs()**
    - Read in and returns 1) the size of the grid, 2) the values in the grid, 3) the number of row indexes for the actions, and 4) the row indexes.

```
void readInputs(int grid[][MAX_SIZE], int *size,
                int indexes[], int *numIndexes)
```

  - **play()**
    - Take in the inputs returned from the readInputs() function.
    - Perform the actions accordingly and return 1) whether the game is completed, 2) the total score (if applicable), 3) the number of blocks left in the grid (if applicable), and 4) the final state of the grid.

```
int play(int grid[][MAX_SIZE], int size,
         int indexes[], int numIndexes,
         int *score, int *numBlocksLeft)
```

Block Buster: readInputs() (5/12)

- How to read inputs?
  - The grid is a 2D array: Similar to Lab #4 Exercise 1 (Easy!)
  - The sequence of row indexes is a 1D array: Similar to Lab #3 Exercise 1 (Easy!)
- How to return all inputs?
  - There is no need to explicitly return the arrays. (Easy!)
  - The size of the grid and the number of row indexes can be return using pointers. (Easy!)

Block Buster: readInputs() (6/12)

```
void readInputs(int grid[][MAX_SIZE], int *size,
                int indexes[], int *numIndexes) {
  int i, j;

  printf("Enter size of grid: ");
  scanf("%d", size);

  printf("Enter grid: \n");
  for (i = 0; i < *size; i++)
    for (j = 0; j < *size; j++)
      scanf("%d", &grid[i][j]);
...
```

Block Buster: readInputs() (7/12)

```
…
  printf("Enter number of indexes: ");
  scanf("%d", numIndexes);

  printf("Enter indexes: ");
  for (i = 0; i < *numIndexes; i++)
  scanf("%d", &indexes[i]);
}
```

Block Buster: play() (8/12)

- How to implement the two actions?
  - Pull: Find the first non-zero element in a row (Intermediate)
  - Push: Find the first element which is equal to or bigger than the weight of the block being pushed (Hard?)
  - Both are similar to Lab 4 Ex #2
- How to alternate between the two actions?
  - If the row index is stored in indexes[i], the action is Pull when i is even, or Push otherwise. (Easy!)
- How to compute the number of blocks left?
  - Count the number of non-zero elements in an array (Easy!)

Block Buster: play() (9/12)

```
int play(int grid[][MAX_SIZE], int size,
         int indexes[], int numIndexes,
         int *score, int *numBlocksLeft){
  int i, j, k, index, block;

  *score = 0;
  *numBlocksLeft = 0;

  … // Code for push and pull

  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
      if (grid[i][j] != 0)
        (*numBlocksLeft)++;

  … // Some other code
```

Block Buster: play() (10/12)

```
   // Code for Push and Pull
   for (i = 0; i < numIndexes; i++){
     index = indexes[i];

     if (index < 0 || index >= size) return 0;

     if (i % 2 == 0){ // Code for Pull
       …
     } else {  // Code for Push
       …
     }
…
```

Block Buster: play() (11/12)

```
// Code for Pull
for (j = 0; j < size; j++){
  if (grid[index][j] != 0){
    block = grid[index][j];
    grid[index][j] = 0;
    break;
  }
}
 if (j == size)
   return 0;
```

Block Buster: play() (12/12)

```
      // Code for Push
      j = -1;
      while (j < size-1 && block > grid[index][j+1])
        j++;
      if (j == -1) return 0;

      for (k = 0; k <= j; k++){
        *score += grid[index][k];
        grid[index][k] = 0;
      }

      grid[index][j] = block;
```

Lessons learn

- Spend time in analysis and design to know
  - which parts are easy
  - which parts are similar to the problems you have solved before
- Code incrementally
  - Compile and test often
  - Always start with the easy parts!
- Practice sufficiently but not blindly
  - Check if you are following the problem solving process
  - Check if you have learnt useful techniques from the exercises
  - Review the programs you have seen / written often

Upcoming challenges…

- Lab 5 deadline: 5 November, 12nn
- Exam: 23 November, 9-11am

End of File