CS1010E Lecture 10

Character Strings and Pointers

Henry Chia (hchia@comp.nus.edu.sg)

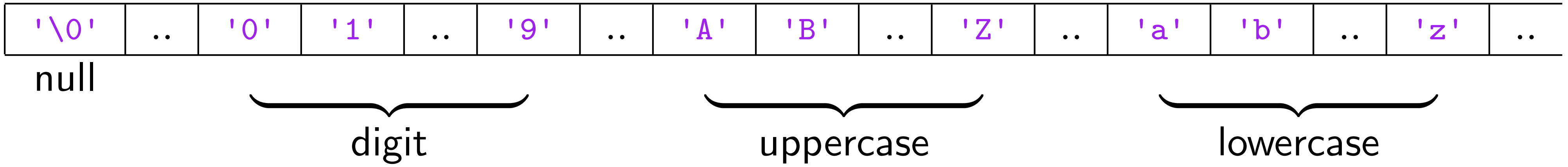
Semester 1 2016 / 2017

Lecture Outline

- Characters
  - Character ordering
  - Declaring variable and arrays
- Strings
  - Declaration
  - Input and output
- Memory address
- Pointer
  - Declaration, arithmetic, assignment, dereference
- String functions

Character Data

- Character value: `'A'`, `'b'`, and `'3'`
- 128 characters ordered according to an ASCII table



`'\0'`	..	`'0'`	`'1'`	..	`'9'`	..	`'A'`	`'B'`	..	`'Z'`	..	`'a'`	`'b'`	..	`'z'`	..

- Character (integral) arithmetic
  - `'A'`+1 $\to$ `'B'` ; `'c'-'a'` $\to$ 2 ; `'9'-'0'` $\to$ 9
  - `('d'-'a')+'A'` $\to$ `'D'` (i.e. case conversion)
- Character relations
  - `'\0'` $<$ `'0'` $<$ `'9'` $<$ `'A'` $<$ `'Z'` $<$ `'a'` $<$ `'z'`
  - `'0'` $\neq 0$ (character digits $\neq$ integer digits)
  - `'\0'` $= 0$

Using Characters

- Declaring character primitives and arrays
  - Uninitialized:
    - `char c, vowels[5];`
  - Initialized:
    - `char c='d';`
    - `char vowels[5]={'a','e','i','o','u'};`
- Output:
  - `printf("The character is %c\n", c);`
- Input:
  - `scanf("%c", &c);`

Reading a Character

- When using `scanf("%c",..)`, every whitespace character (space, tab, enter, ..) is read as a separate character
- What happens when the following program is executed?

```
#include <stdio.h>

int main(void) {
   char c1, c2;

   printf("Enter first character: ");
   scanf("%c", &c1);
   printf("The first character is %c\n", c1);

   printf("Enter second character: ");
   scanf("%c", &c2);
   printf("The second character is %c\n", c2);

   return 0;
}
```

Character String

- **Character string** — character array containing a sequence of characters terminated with a null `'\0'` (or 0)
- Character string constants are enclosed in double quotes, as in `"cs1010e"`, `"E"`, and `"1010"`
- Initialize a character string (declaration only)
  - `char str[8]="cs1010e";`
- To store a string of n characters, ensure an array size of at least $n+1$ to store the null character `'\0'`
- Output a character string using %s

```
printf("Module code: %s\n", str);
```

- %s prints until the **first occurrence** of `'\0'`

Reading a Word

- Use %s to read a string as a word delimited by whitespaces

```
#include <stdio.h>

int main(void) {
    char word1[40], word2[40];

    printf("Enter first word: ");
    scanf("%s", word1); /* & operator not needed */
    printf("The first word is %s\n", word1);

    printf("Enter second word: ");
    scanf("%s", word2);
    printf("The second word is %s\n", word2);

    return 0;
}
```

- In the above, ensure that each word is no longer than 39 characters since `scanf` appends a `'\0'` at the end

Reading a Line

- Read an entire line (including spaces)

```c
#include <stdio.h>

void readLine(char str[]);

int main(void) {
   char str[100]; /* read up to 99 chars + '\0' */

   readLine(str);
   printf("%s\n", str);
   return 0;
}

void readLine(char str[]) {
   int i = 0;

   scanf("%c", &(str[i]));
   while (str[i] != '\n') { /* read until newstr */
      i=i+1;
      scanf("%c", &(str[i]));
   }
   str[i] = '\0'; /* terminate the string */
   return;
}
```

String as Function Argument

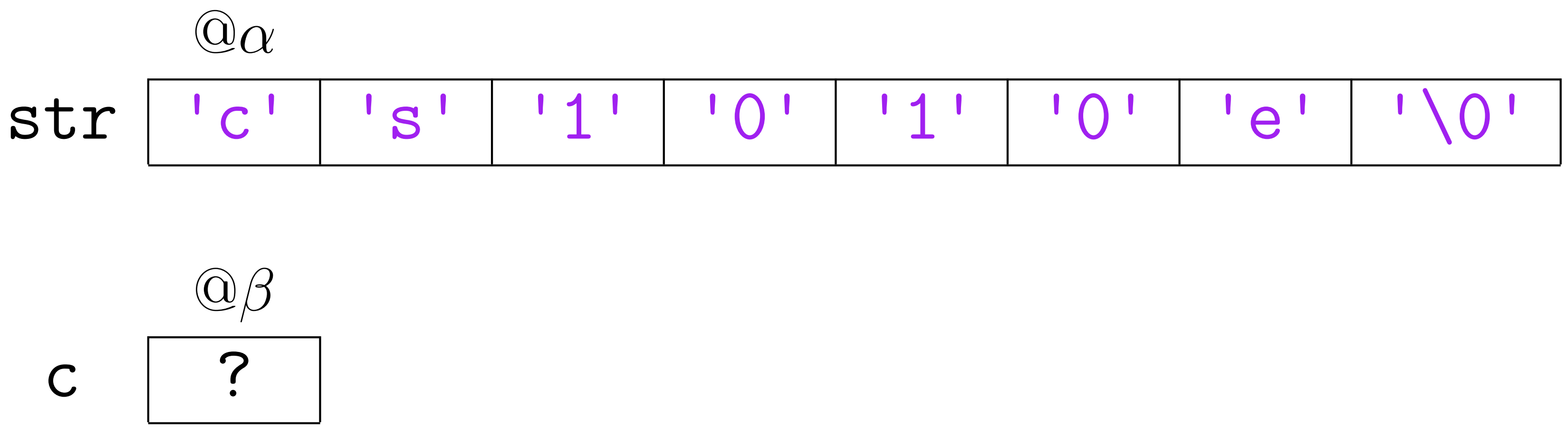
- Since `'\0'` terminates the string `str`, no need to pass the number of characters in `str` to the search function

```c
int search(char str[], char c);

int main(void) {
   char str[8]="cs1010e", c;

   printf("Enter a character: ");
   scanf("%c", &c);
   printf("Finding %c in %s returns index %d\n", c, str, search(str,c));
   return 0;
}

int search(char str[], char c){
   int i = 0;

   while ((str[i] != '\0') && (str[i] != c)) {
      i++;
   }
   if (str[i] == '\0') {
      return -1;
   } else {
      return i;
   }
}
```

Memory Addresses

- Memory locations are assigned to variables when a program is executed
- A memory location is uniquely defined by an **address** value
- Actual addresses allocated to variables are determined each time the program is executed

```
char str[8]="cs1010e", c;
```



Address Operator

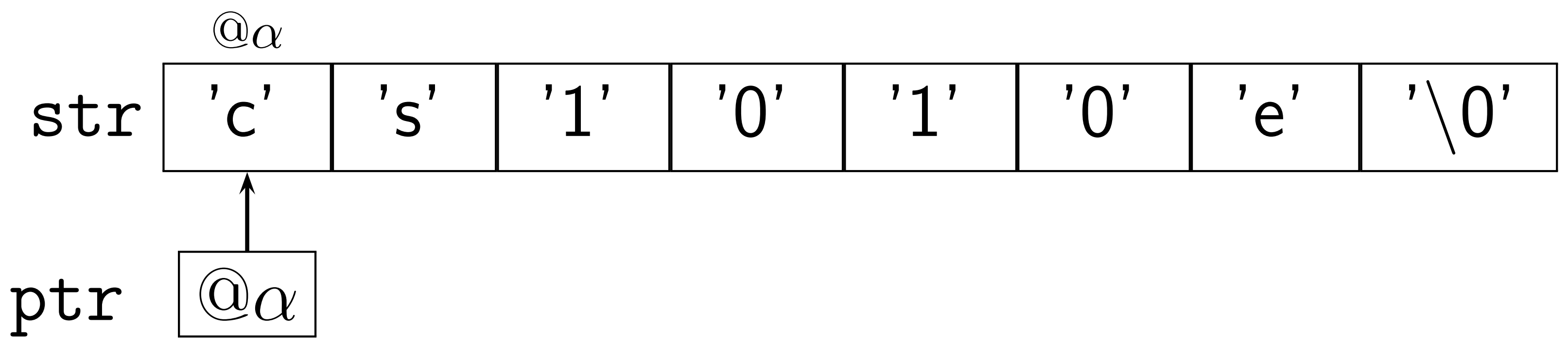
```
char str[8]="cs1010e", c;

printf("Address of variable c is %d\n", &c);
printf("Constant address of array str is %d\n", str);
printf("Address of str[0] is %d\n", &(str[0]));
```

- The address-of operator `&` is used to obtain the address of a primitive variable, e.g. `&c`
- An array variable is a constant address and is equivalent to the address of the first element of the array
- An address value can be stored in a pointer variable

  ```
  char str[8]="cs1010", c;
  char *ptr; /* ptr declared as pointer to char */

  ptr = &c;  /* ptr stores addr of variable c */
  ptr = str; /* equivalent to ptr = &(str[0]) */
  ```

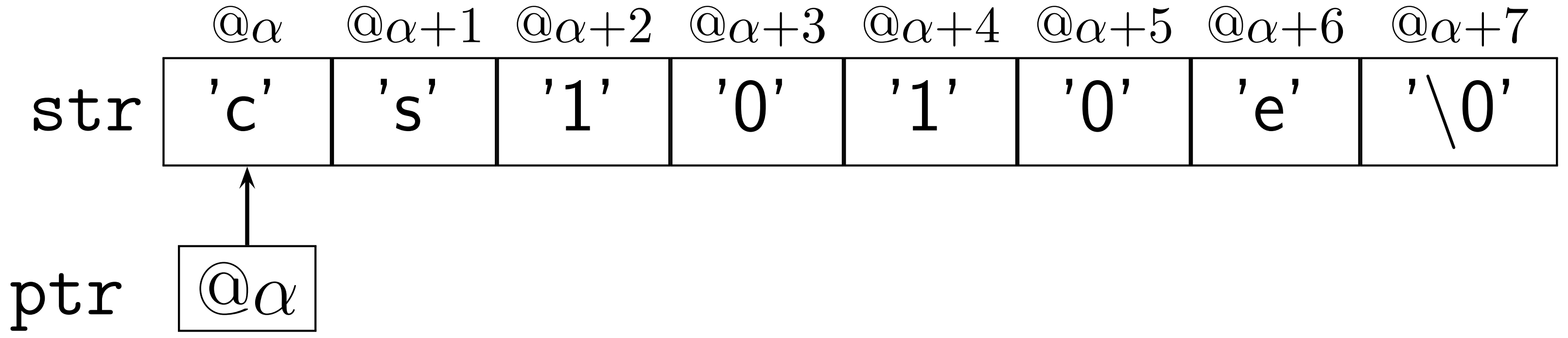
Dual Use of `[]` and `*`

```
char str[8]="cs1010e"; /* str[8] declares array of 8 chars */
char *ptr;             /* *ptr declares ptr as pointer */

ptr = str;
printf("%c", str[0]); /* str[0] refers to 0th element of str */
printf("%c", *ptr);   /* *ptr refers to value that ptr points */
```



- Distinguish the use of [] and ∗ within
  - declarations — allocate memory for arrays or pointers
  - statements — indexing/subscripting or dereferencing

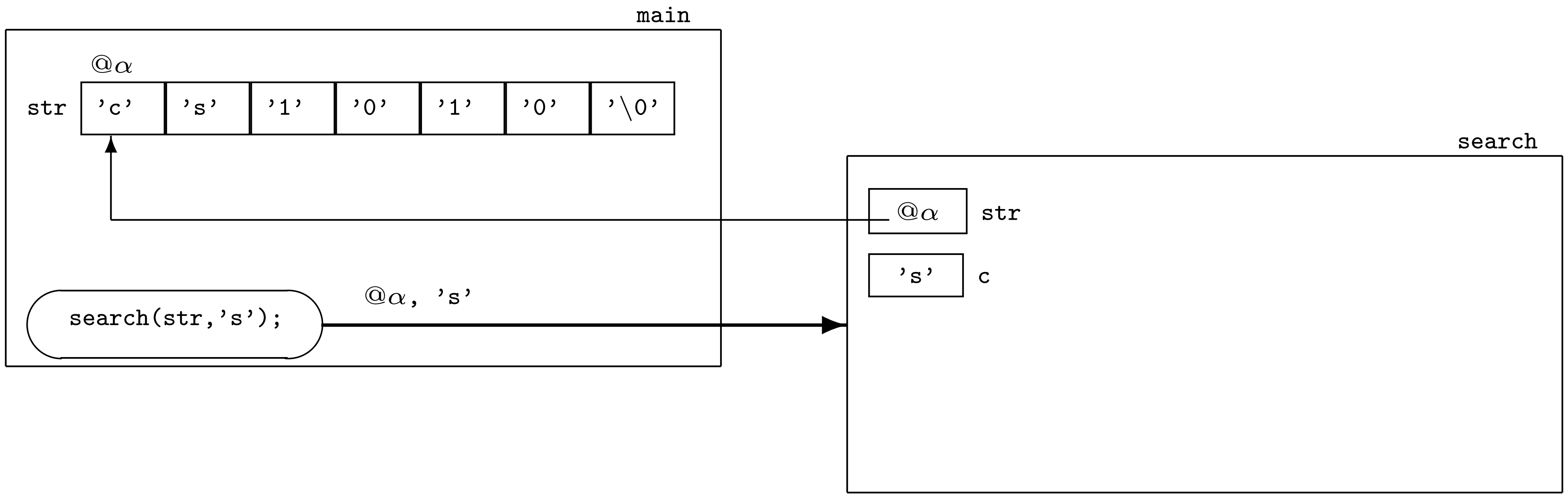
Pointer Arithmetic



- `str[1]` is the second element of str, or one element to the right of `str[0]`
- Likewise, `(ptr + 1)` is one element to the right of `ptr` and `*(ptr + 1)` refers to that element
- So `str[1]` is equivalent to `*(ptr+1)`

Pointer as Function Parameter

- Pass-by-address-value: when the array `str` is passed to function `search`, the address value `&(str[0])` is passed
- The parameter `str` in function `search` takes a copy of the address of the array (@α) passed by the caller function `main`



Pointer as Function Parameter

- Parameter str in search can be re-defined as
  `int search(char *str, char c)`
- Parameter declarations `char *str` or `char str[]` declares str as a pointer that stores an address value
- Dereferencing `*` can be used in place of subscripting `[]`

```
int search(char str[], char c) {
   int i = 0;

   while ((str[i] != '\0') &&
          (str[i] != c)) {
      i++;
   }
   if (str[i] == '\0') {
      return -1;
   } else {
      return i;
   }
}
```

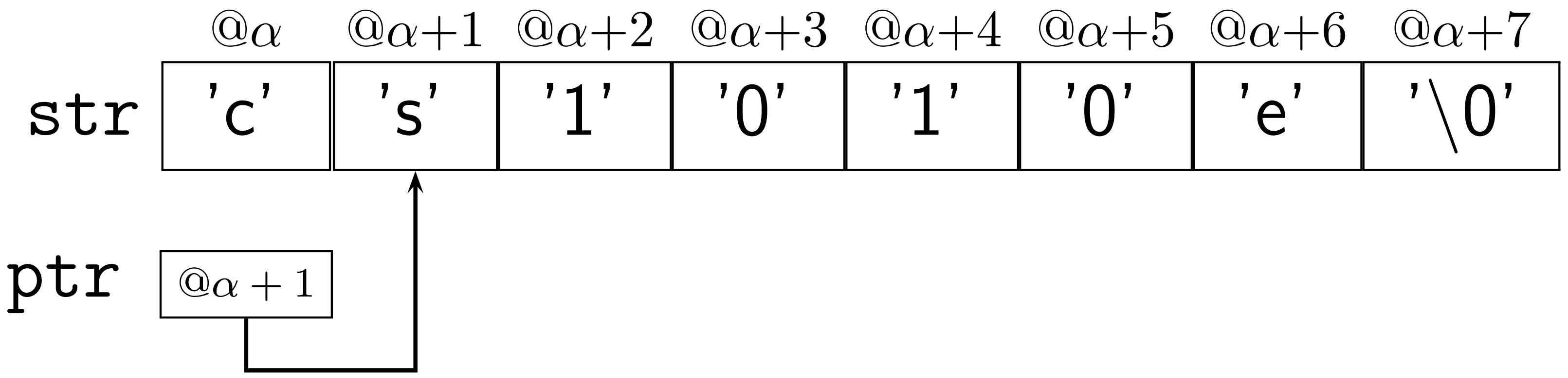
```
int search(char *str, char c) {
   int i = 0;

   while ( (*(str + i) != '\0') &&
           (*(str + i) != c) ) {
      i++;
   }
   if (*(str + i) == '\0') {
      return -1;
   } else {
      return i;
   }
}
```

Pointer Assignment

- A pointer can be assigned with a different address value

  ```
  char str[8]="cs1010e";
  char *ptr;

  ptr = str;
  ```

- Make `ptr` point to `str[1]`: `ptr=ptr+1; /* or ptr++ */`



- Now, `*ptr` is the character `'s'`
- To zero a pointer (not pointing anywhere), simply `ptr = 0`

Pointer Assignment

```c
int search(char *str, char c) {
   char *ptr;

   ptr = str;
   while ( (*ptr != '\0') && (*ptr != c) ) {
      ptr++;
   }
   if (*ptr == '\0') {
      return -1;
   } else {
      return ptr - str;
   }
}
```

- Addresses can be subtracted, e.g. in the above, `ptr` is `(str - ptr)` elements to the right of `str`
- However, addresses cannot be added; it does not make sense

Returning Addresses

- An address can be returned from a function
- The search function below returns the address of the element in str where c is found; or the address of the terminating '\0' in the case of an unsuccessful search

```
#include <stdio.h>

char *search(char *str, char c);

int main(void) {
   char str[8]="cs1010e", c;
   char *ptr;

   printf("Enter a character: ");
   scanf("%c", &c);
   ptr = search(str,c);
   printf("Substring starting with %c in ", c);
   printf("%s is %s\n", str, ptr);
   return 0;
}
```

```
char *search(char *str, char c) {
   while ( (*str != '\0') &&
           (*str != c) ) {
      str++;
   }
   return str;
}
```

String Functions

- The Standard C library contains a number of string functions
- Usage of these functions requires

  ```c
  #include <string.h>
  ```

- Focus on four functions:

  — `strlen`, `strcmp`, `strcpy` and `strcat`
- `unsigned int strlen(const char *s);`
  - Returns the length of the string `s`, e.g.

    ```c
    char str[8] = "cs1010e";
    int n;

    n = (int)strlen(str);
    ```

- `const` keyword prevents the string `s` from being modified

String Functions

- `int strcmp(const char *s, const char *t)`
  - Compares each `s`[i] and `t`[i], $i = 0, 1, 2, \ldots$
  - A negative value is returned if there exists a `s`[i]<`t`[i] with `s`[j]=`t`[j], $\forall j < i$
  - A positive value is returned if there exists a `s`[i]>`t`[i] with `s`[j]=`t`[j], $\forall j < i$
  - zero is returned if `s`[i]=`t`[i], $\forall i < k$ and `s`[k]=`t`[k]=`'\0'`
- Examples:
  - `strcmp("cs1010e","cs1010s")` returns a number < 0
  - `strcmp("cs1010e","cs1010")` returns a number > 0
  - `strcmp("cs1010e","cs1010e")` returns exactly 0

String Functions

- `char *strcpy(char *s, const char *t)`
  - Copies string `t` to string `s`
- `char *strcat(char *s, const char *t)`
  - Concatenates (joins) string `t` to the end of string `s`
- Ensure that `s` has sufficient space to accommodate `t`

```
char str1[10]="cs", str2[10]="1010";
strcat(str2,"e");  /* str2 is now "1010e" */
strcat(str1,str2); /* str1 is now "cs1010e" */
printf("%s %s\n", str1, str2);
```

- `strcpy`/`strcat` returns pointer to the result string; this allows string functions to be composed:

```
char str1[10]="cs", str2[10]="1010";
int n = (int)strlen(strcat(str1,strcat(str2,"e")));
```

Problem Solving: igPay atinLay

- To convert an English word to *Pig Latin*,
  1. Find the first vowel in the word
  2. The sub-string beginning at the start till just before the first vowel is moved to the end of the word
  3. Add "ay" at the end of the word

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

bool isVowel(char c);
char *piglatin(char *out, char *in);

int main(void) {
   char in[40], out[40];

   printf("Enter word: ");
   scanf("%s", in);
   printf("%s\n", piglatin(out,in));
   return 0;
}
```

Problem Solving: igpay atinlay

```
bool isVowel(char c) {
   char vowels[11]="aeiouAEIOU";
   int i = 0;

   while (i < 10 && c != vowels[i]) {
      i++;
   }
   return i < 10;
}

char *piglatin(char *out, char *in) {
   char *v;

   v = in;
   while (*v != '\0' && !isVowel(*v)) {
      v++;
   }
   strcpy(out,v);
   *v = '\0';
   return strcat(strcat(out,in),"ay");
}
```

Lecture Summary

- Characters and Strings
  - Understanding character ordering to perform conversions
  - Strings are character arrays with a terminating `'\0'`
  - Operation of string functions are dependent on the position of `'\0'`
- Pointers
  - Addresses are passed to functions and stored in pointer variables via **pass-by-address-value**
  - Pointer assignment and dereferencing apply to both primitive variables and array elements
  - Pointer arithmetic applies to only array elements