Part III
Strings

1) Introduction
2) Definition
3) Using Strings
4) Input/Output
5) Using “string.h” Library
6) Strings as Function Parameters
7) Complex example
8) Laboratory example
3.1 Introduction

- **String:**
  - A one-dimensional array with items of type character ("char")

- **Idea:**
  
<table>
<thead>
<tr>
<th>char</th>
<th>char</th>
<th>char</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>h</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>r</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>N</td>
</tr>
</tbody>
</table>

- **Examples:**
  - String of letters
    
    | ‘S’ | ‘t’ | ‘r’ | ‘i’ | ‘n’ | ‘g’ |
  
  - String of different alphanumeric characters
    
    | ‘9’ | ‘1’ | ‘4’ | ‘f’ | ‘7’ |

- **Components:**
  - String name
  - Number of items
  - Items’ order
3.2 Definition

"char" <string_name> `[<string_size>]`

- **Components:**
  - Name:
    
    `<string_name> = <array_name>`
  - Maximum number of items (array size):
    
    `<string_size> = <array_size>`
  - Order of items (implicit)
    
    from 0 (first item) to `<string_size>-1` (last item)

- **Effect:**
  - Computer allocates a contiguous memory block of size
    `<string_size> * sizeof("char")`

- **Usage:**
  - `char name[5];` \(\Rightarrow 1 \times 5 = 5\) B
  - `char text[255];` \(\Rightarrow 1 \times 255 = 255\) B
  - `char str[1000];` \(\Rightarrow 1 \times 1000 = 1000\) B
• Memory allocation:

- The contiguous memory block consists of memory locations allocated as each string item had been allocated separately

- The size of each such memory location is equal to the size required to store a variable of type “char”

- An extra memory location is added at the end of the string and when using string-processing functions will be initialised with a special character “NULL”: ‘0’

• Notes:

- All the observations made for arrays apply also to strings

- Strings can be processed at the level of their items (characters) in similar fashion any array is

- Apart from this, a special library (“string.h”) was built that provides string-processing functions
3.3 Using Strings

- **String level**
  \(<string\_name>\)

- **Character level**
  \(<string\_name> \[ \`` <index> \`` \] \)

- **Example:**
  - name[0]
  - name
  - text[10]

- **Notes:**
  - Although allocated, it is not required to use every location within a string
    - however the locations used are usually contiguous and end with the “NULL” character (“\0”)
  - Regular operations with strings include:
    - input/output and parsing/processing
3.4 Input/Output Operations

• String initialisation:
  • At declaration
    - character-by-character
      e.g.
      ```
      char name[5] = {'J','o','h','n','\0'};
      ```
    - as a string
      e.g.
      ```
      char name[5] = "John";
      ```
  • During execution
    - with user input
      - character-by-character
        e.g.
        ```
        char name[5], ch;
        for (index = 0; index < 5; index++)
        scanf("%c%c", &name[index], &ch);
        name[index] = '\0';
        ```
      - as a string
        e.g. 1
        ```
        char name[5];
        scanf("%s", name);
        ```
        e.g. 2
        ```
        char name[5];
        gets(name);
        ```
- without user input

- character-by-character

  e.g.
  
  char name[5];
  for (index = 0; index < 5; index++)
      name[index] = ‘a’ + index;

  name[index] = ‘\0’;

- as a string

  e.g. 1
  
  char name[5];
  strcpy(name, “John”);

  e.g. 2
  
  char name1[5];
  char name2[5];
  strncpy(name2, name1, 3);

• Input Functions:

  #include <stdio.h>

  char name[20];

  scanf(“%s”, name);

  gets(name);

• Notes:

  • “scanf” stops reading when encounter a “blank” character (“space”)

  • “gets” reads all the characters, including “blank” characters
• String Output:

- character-by-character

e.g.
char name[5];

for(index = 0;
    index < 5 && name[index] != '\0';
    index++)
    printf("%c", name[index]);

- as a string

e.g. 1
char name[5];
printf("%s", name);

   e.g. 2
char name[5];
puts(name);

• Output Functions:

#include <stdio.h>
char name[20];

printf("%s", name);
puts(name);
• **Example:**

  • Direct customers to counter #8 if their name starts with a letter before ‘M’ in alphabetical order and to counter #9 otherwise

    ```c
    #include <stdio.h>
    #include <stdlib.h>
    int main()
    {
        char name[20];
        printf("Enter your name:");
        scanf("%s", name);

        if (name[0] < 'M')
            printf("%s – counter #8", name);
        else
            printf("%s – counter #9", name);

        return (EXIT_SUCCESS);
    }
    ```

• **Problem:**

  This solution does not check if the first character of the name typed in by the user is a letter.

  Ex: 0John
3.5 The ASCII Character Set

- **ASCII** = American Standard Code for Information Interchange”

- **ASCII Table** = A set of codes used to convert letters (and other "characters") to and from numbers so that they can be stored inside the computer and manipulated by the computer

  - Every letter, number, punctuation mark, etc. (known as a character) has assigned a unique code (value) according to its position within the ASCII table

- The standard **ASCII table is a 7-bit code and defines 128 character codes** (from 0 to 127)

  - the first 32 characters are **non-printing control characters** (e.g. Carriage return, Line feed, Escape)

  - the remaining 96 character codes are **representable characters**

- Nowadays computers work with 8-bit bytes => can represent 256 different values

  - In addition to the 128 standard ASCII codes there are other 128 codes, known as **extended ASCII**, that are platform- and locale-dependent.

    - **ANSI extend ASCII** used by MS Windows & UNIX operating systems

    - **OEM extend ASCII** used under the old DOS operating system
- **ISO Latin-1 extend ASCII** that contains various accented characters (e.g. È, ä) and other letters (e.g. ß,æ) used for writing languages of Western Europe as well as some special characters.

### ASCII Character Set

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>!</td>
</tr>
<tr>
<td>33</td>
<td>&quot;</td>
</tr>
<tr>
<td>34</td>
<td>#</td>
</tr>
<tr>
<td>35</td>
<td>$</td>
</tr>
<tr>
<td>36</td>
<td>%</td>
</tr>
<tr>
<td>37</td>
<td>&amp;</td>
</tr>
<tr>
<td>38</td>
<td>'</td>
</tr>
<tr>
<td>39</td>
<td>(</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>*</td>
</tr>
<tr>
<td>43</td>
<td>+</td>
</tr>
<tr>
<td>44</td>
<td>,</td>
</tr>
<tr>
<td>45</td>
<td>-</td>
</tr>
<tr>
<td>46</td>
<td>.</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>5</td>
</tr>
<tr>
<td>54</td>
<td>6</td>
</tr>
<tr>
<td>55</td>
<td>7</td>
</tr>
<tr>
<td>56</td>
<td>8</td>
</tr>
<tr>
<td>57</td>
<td>9</td>
</tr>
<tr>
<td>58</td>
<td>:</td>
</tr>
<tr>
<td>59</td>
<td>;</td>
</tr>
<tr>
<td>60</td>
<td>&lt;</td>
</tr>
<tr>
<td>61</td>
<td>=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td>&gt;</td>
</tr>
<tr>
<td>63</td>
<td>?</td>
</tr>
<tr>
<td>64</td>
<td>@</td>
</tr>
<tr>
<td>65</td>
<td>A</td>
</tr>
<tr>
<td>66</td>
<td>B</td>
</tr>
<tr>
<td>67</td>
<td>C</td>
</tr>
<tr>
<td>68</td>
<td>D</td>
</tr>
<tr>
<td>69</td>
<td>E</td>
</tr>
<tr>
<td>70</td>
<td>F</td>
</tr>
<tr>
<td>71</td>
<td>G</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>H</td>
</tr>
<tr>
<td>73</td>
<td>I</td>
</tr>
<tr>
<td>74</td>
<td>J</td>
</tr>
<tr>
<td>75</td>
<td>K</td>
</tr>
<tr>
<td>76</td>
<td>L</td>
</tr>
<tr>
<td>77</td>
<td>M</td>
</tr>
<tr>
<td>78</td>
<td>N</td>
</tr>
<tr>
<td>79</td>
<td>O</td>
</tr>
<tr>
<td>80</td>
<td>P</td>
</tr>
<tr>
<td>81</td>
<td>Q</td>
</tr>
</tbody>
</table>

The binary number for 'A' is **1000001**.
<table>
<thead>
<tr>
<th>Code</th>
<th>82</th>
<th>83</th>
<th>84</th>
<th>85</th>
<th>86</th>
<th>87</th>
<th>88</th>
<th>89</th>
<th>90</th>
<th>91</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>92</th>
<th>93</th>
<th>94</th>
<th>95</th>
<th>96</th>
<th>97</th>
<th>98</th>
<th>99</th>
<th>100</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>102</th>
<th>103</th>
<th>104</th>
<th>105</th>
<th>106</th>
<th>107</th>
<th>108</th>
<th>109</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>112</th>
<th>113</th>
<th>114</th>
<th>115</th>
<th>116</th>
<th>117</th>
<th>118</th>
<th>119</th>
<th>120</th>
<th>121</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>122</th>
<th>123</th>
<th>124</th>
<th>125</th>
<th>126</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>z</td>
<td>{</td>
<td></td>
<td>}</td>
<td>~</td>
</tr>
</tbody>
</table>
• **Previous Example:** Direct customers to counter #8 if their name starts with a letter before ‘M’ in alphabetical order and to counter #9 otherwise

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char name[20];
    printf("Enter your name:");
    scanf("%s", name);

    if ((name[0] >= 'A' && name[0] < 'M') ||
        (name[0] >= 'a' && name[0] < 'm'))
        printf("%s – counter #8", name);
    else
        if((name[0] >= 'M' && name[0] <= 'Z') ||
           (name[0] >= 'M' && name[0] <= 'z'))
            printf("%s – counter #9", name);
        else
            printf("ERROR: Incorrect name");
    return (EXIT_SUCCESS);
}
```

• **A simpler solution**

```c
#include <ctype.h>
int main()
{

    char letter;
    letter = toupper(name[0]);

    if (letter >= 'A' && letter < 'M')
        printf("%s – counter #8", name);
    else
        if (letter >= 'M' && letter <= 'Z')
            printf("%s – counter #9", name);
        else
            printf("ERROR: Incorrect name");
    return (EXIT_SUCCESS);
}
```

**ctypes.h library**

- int toupper( int ch )
- int tolower( int ch )
- int isalpha( int ch )
- int isdigit( int ch )
3.6 Using “string.h” Library

• Notes:
  • “string.h” library contains a large set of functions that can process strings
  • “string.h” must be included in the program before any of its functions is used
  • strings must end with ‘\0’ in order for the functions to be able to process them

• Most Important Functions:
  • Comparison between two strings:
    ```c
    int strcmp(char* string1, char* string2);
    ```
    The function `strcmp()` returns:
    - a positive integer if string `string1` is lexically greater than string `string2`;
    - zero if the two strings are identical;
    - a negative integer if string `string1` is lexically less than string `string2`

    ```c
    int strncmp(char* string1, char* string2, int n);
    ```
    The comparison is performed only on `n` characters

  • Copying a string into another string:
    ```c
    char* strcpy(char* dest, const char* src);
    ```
    Copies the content of `src` to `dest` until ‘\0’ is reached. `dest` should be large enough to contain the `src` string
    The function returns the `dest` string

    ```c
    char* strncpy(char* dest, char* src, int n);
    ```
    Only `n` characters are copied
• Appending a string to another string:

```c
char* strcat(char* dest, const char* src);
```

Appends the content of `src` to `dest` until `\0` is reached.
`dest` should be large enough to contain both `dest` and `src`.
The function returns the `dest` string.

```c
int strncat(char* dest, char* src, int n);
```

Only `n` characters are appended.

• Determining the length of a string:

```c
int strlen(char* string);
```

Returns the length of `string` not including `\0`.

• Examples:

• Prints the first five letters of a text

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

int main ()
{
    char str1[] = "To be or not to be";
    char str2[6];

    strncpy(str2,str1,5);
    str2[5] = '\0';

    puts(str2);
    return(EXIT_SUCCESS);
}
```

• Output: To be
• Compare a user entered password with a keyword and block the access if not correct

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

int main ()
{
    char keyword[] = "CD_U.692";
    char password[10];
    do
    {
        printf("Enter password:");
        gets(password);
    } while(strcmp(password, keyword) != 0);
    printf("Welcome!
");
    return(EXIT_SUCCESS);
}
```

• Determine the length of an user-inputted text

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

int main ()
{
    char text[100];
    printf("Enter a text: ");
    gets(text);
    printf("Sentence entered is:\n %s", text);
    printf("\nIt has %d chars", strlen(text));
    return(EXIT_SUCCESS);
}
```
3.7 Strings as Function Parameters

• Notes:
  - Strings, as arrays, are passed as parameters to any function using “by-address” strategy
  - In consequence functions receive only the reference to the beginning of the contiguous memory zone allocated for the array
  - As there is a ‘\0’ that indicates the end of the string, there is no need to indicate explicitly the length of the string via another parameter

• Example:
  - Function that prints the letters of a text, letter-by-letter:

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

int main ()
{
    char text[100];
    printf("Enter a text: ");
    gets(text);
    print_letters(text);
    return(EXIT_SUCCESS);
}
```

```c
/* void print_letters(char text[]) */
void print_letters(char* text)
{
    int i;

    for (i = 0; i < strlen(text); i++)
        printf("%d-th letter=%c\n",i, text[i]);
}
```

Output:
Enter a text: john
0-th letter=j
1-th letter=o
2-th letter=h
3-th letter=n
3.8 Complex Example

• Example:
  • Maintain the record of average marks for the students from a school given their studentIDs and Names. Allow for printing separately students whose names start with a certain letter
  • Assume that functions search_student() and set_marks() are provided
  • Modify functions init_records() and list_marks() according to the problem requirements

#include <stdio.h>
#include <stdlib.h>

/* declarations */
/* function that searches for the record position given a studentID and returns it */
int search_student (int sIDs[], int studID, int size);

/* function that sets the average mark in the records given a position */
void set_mark (float avgM[], int pos, float avgmark);

/* function that initialises student records and returns the number of registered students */
int init_records_letter (int sIDs[], float avgM[], char sNames[][20], int maxsize);
/* function that lists the studentIDs and average marks for the registered students whose names start with the indicated letter*/
void list_marks_letter (int sIDs[], float avgM[],
                       char sNames[][20], int size, char letter);

/* main function */

int main()
{
    /* 1-dimensional array that stores studentIDs for all registered students */
    int studentIDs[5000];

    /* 1-dimensional array that stores average marks for all registered students */
    float avgMarks[5000];

    /* 1-dimensional array of strings (2-D array) that stores all registered students names */
    char sNames[5000][20];

    /* no current registered students (no records) */
    int no_recs = 0;

    /* Other variables */
    int studID;
    float mark;
    char letter, ch;

    /* Welcome message */
    printf("Welcome to the program!\n");

    /* Init. records by calling init_records() */
    no_recs = init_records_letter (studentIDs,
                                   avgMarks, sNames, 5000);

    /* Ask user for input data */
    printf("Enter student ID: ");
    scanf("%d", &studID);
printf("Enter the mark: ");
scanf("%f%c", &mark,&ch);

/* Update the records with user data */
set_mark (avgMarks,
    search_student(studentIDs, no_recs,
    studID),mark);

/* Ask for a letter */
printf("Enter the letter: ");
scanf("%c%c", &letter, &ch);

/* Print studentIDs and students marks only for students whose names start with given letter */
list_marks_letter (studentIDs, avgMarks,
    sNames, no_recs, letter);

printf("Goodbye!
");

return (EXIT_SUCCESS);
}

/* function definitions */
void set_mark (float avgM[], int pos, float avgMark)
{
    if (pos >= 0)
    {
        avgM[pos] = avgMark;
    }
}
int search_student (int sIDs[], int size, int studID)
{
    int i;
    
    for (i = 0; i < size; i++)
        if (sIDs[i] == studID)
            return i;

    return -1;
}

int init_records_letter (int sIDs[], float avgM[],
                        char sNames[][20], int maxsize)
{
    int size = 0;
    int ans = 1;
    char name[20], ch;
    
    while (ans != 0 && size < maxsize)
    {
        printf("New student registration: \n");

        /* Read student ID */
        printf("Enter student ID: \n");
        scanf("%d", &sIDs[size]);

        /* Initialise average mark with 0.0 */
        avgM[size] = 0.0;

        /* Read student name */
        printf("Enter student name: \n");
        scanf("%s", sNames[size]);

        /* Update no of records */
        size++;
    }
/* Check for new student registration */
printf("Continue (yes=1, no=0):
\n");
scanf("%d", &ans);
}
return size;


}  

*/ Check for new student registration */
printf("Continue (yes=1, no=0):
\n");
scanf("%d", &ans);
}
return size;


}  

void list_marks_letter (int sIDs[], float avgM[], char sNames[][20], int size, char letter)
{
    int i;

    printf("| Name  | StudID | Avg Mark |
\n");

    for (i = 0; i < size; i++)
        if (sNames[i][0] == letter)
            printf("|%8s|%8d|%8.2f|\n",
                sNames[i], sIDs[i], avgM[i]);


}  

NOTE:

Modify the example such as it allows for multiple modifications of marks and printing of records, respectively
Main tasks:

- Initialise the records (studentIDs, avgMarks, sNames arrays)
  - Read each student ID
  - Place 0.0 in avgMarks for each student
  - Read each student name as a string
- Read one student ID and the average mark
- Update the corresponding record in avgMarks for that student
- Print student IDs and marks only for students whose names start with a given letter
### 3.9 Laboratory Exercises

- Cryptography

#### Caesar’s Cipher

**Idea:**
Original letter + constant => Ciphered letter

**Example:**
- constant = 4
- ‘a’ => ‘e’
- ‘m’ => ‘q’
- ‘z’ => ‘d’

#### Vigenere Cipher

**Idea:**
Original letter + constant[i] => Ciphered letter,
i = 0, 1, 2, ..., N

**Example:**
- N = 3
- ‘a’ => ‘k’
- ‘m’ => ‘p’
- ‘z’ => ‘e’
- ‘b’ => ‘l’