

The background features a white canvas with several large, flowing, abstract shapes in shades of light green, light blue, and light purple. Interspersed among these swirls are numerous small, yellow, triangular shapes pointing in various directions, creating a dynamic and celebratory feel.

Data Structure & Algorithm Basic Lab – week 1

Introduction

- C Programming practice in UNIX environment.
- Programming topics related to [Data Structures and Algorithms]
- Compiler: gcc
- Editor: Emacs, K-Developer.

gcc syntax

- Parameter:

- Wall : turn on all alerts

- c: make object file

- o: name of output file

- g: debug information

- l: library

```
gcc -Wall hello.c -o runhello
```

```
./runhello
```

The background features a light blue and white pattern of overlapping circles and streamers. On the left side, there are three balloons: a light green one at the top, a light blue one in the middle, and a light purple one at the bottom. Yellow streamers and triangular flags are scattered throughout the scene.

This week: Basic Data Structures and Algorithms

- Topic:
 - Array, String, Pointer Review
 - Character based File operations in UNIX
 - Programming Exercises

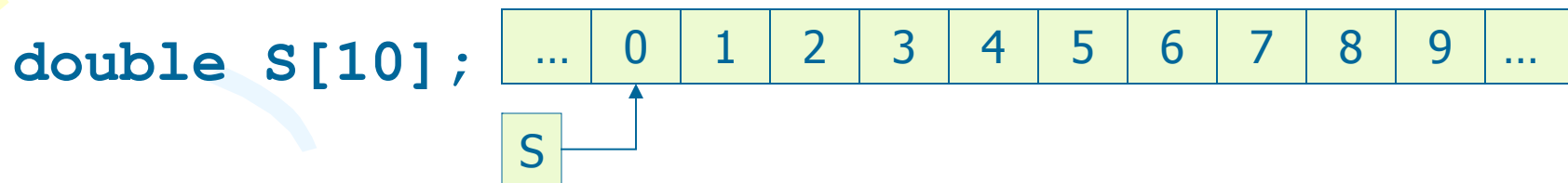
Array

- A block of many variables of the same type
- Array can be declared for any type
 - E.g. `int A[10]` is an array of 10 integers.
- Examples:
 - list of students' marks
 - series of numbers entered by user
 - vectors
 - matrices

Arrays in Memory

- Sequence of variables of specified type
- The array variable itself holds the address in memory of beginning of sequence

- Example:



- The k-th element of array A is specified by `A[k-1]` **(0-based)**

Example - reverse

```
#include <stdio.h>

int main(void)
{
    int i, A[10];

    printf("please enter 10 numbers:\n");
    for(i=0; i<10; i++)
        scanf("%d", &A[i]);

    printf("numbers in reversed order:\n");
    for(i=9; i>=0; i--)
        printf("%d\n", A[i]);

    return 0;
}
```

Exercise

- Write a program that gets an input line from the user (ends with '\n') and displays the number of times each letter appears in it.

The output for the input line: "hello, world!"

The letter 'd' appears 1 time(s).

The letter 'e' appears 1 time(s).

The letter 'h' appears 1 time(s).

The letter 'l' appears 3 time(s).

The letter 'o' appears 2 time(s).

The letter 'r' appears 1 time(s).

The letter 'w' appears 1 time(s).

Assume all inputs are lower-case!

Solution

```
#define ALPHABET_LEN 26
```

```
int main(void)
```

```
{  
    int i = 0,  
    count[ALPHABET_LEN] = {0};  
    char c = '\0';  
  
    printf("Please enter a line of text: \n");
```

```
    /* Read in letter by letter and update the count array  
    */
```

```
    c = getchar();  
    while (c != '\n' && c >= 0)  
    {  
        if (c <= 'z' && c >= 'a')  
            ++count[c - 'a'];  
        if (c <= 'Z' && c >= 'A')  
            ++count[c - 'A'];  
        c = getchar();  
    }
```

Solution

```
for (i = 0; i < ABC_LEN; ++i) {  
    if (count[i] > 0)
```

```
        printf("The letter '%c' appears %d time(s).\n", 'a' + i,  
            count[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

The background features a light blue and white color scheme with decorative elements including a green balloon in the top left, a purple balloon in the bottom left, and several yellow streamers and triangular shapes scattered throughout.

Exercise (20 minutes)

- Implement a function that accepts two integer arrays and returns 1 if they are equal, 0 otherwise
- Write a program that accepts two arrays of integers from the user and checks for equality

Solution

```
#include <stdio.h>
```

```
#define SIZE 5
```

```
int compare_arrays(int arr1[], int arr2[], int size)
```

```
{
```

```
    int i = 0;
```

```
    for (i = 0; i < size; ++i)
```

```
    {
```

```
        if (arr1[i] != arr2[i])
```

```
            return 0;
```

```
    }
```

```
    /* if we got here, both arrays are identical */
```

```
    return 1;
```

```
}
```

Solution

```
int main(void)
{
    int input1[SIZE], input2[SIZE], i;

    printf("Please enter a list of %d integers:\n",
        SIZE);
    for (i = 0; i < SIZE; ++i) scanf("%d", &input1[i]);

    printf("Please enter another list of %d
    integers:\n", SIZE);
    for (i = 0; i < SIZE; ++i) scanf("%d", &input2[i]);

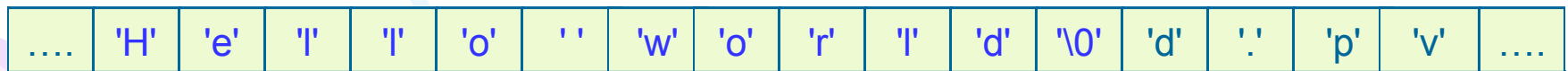
    if (compare_arrays(input1, input2, SIZE) == 1)
        printf("Both lists are identical!\n");
    else
        printf("The lists are not identical...\n");

    return 0;
}
```

Strings

- An array of characters
- Used to store text
- Another way to initialize:

```
char str[] = "Text";
```



Terminator

String

- In order to hold a string of N characters we need an array of length $N + 1$
- So the previous initialization is equivalent to

```
char str[] = {'b', 'l', 'a', 'b', 'l',  
'a', '\0'};
```

String and character related function

- `getchar()`
 - `c = getchar();`
- `scanf`
 - `scanf("%s", str);`
- `gets()`
 - `gets(str);`

String and character related function

- `strlen(const char s[])`
returns the length of s
- `strcmp(const char s1[],
const char s2[])`
compares s1 with s2
- `strcpy(char s1[],
const char s2[])`
copies to contents of s2 to s1

Exercise

- write a function that:
 - gets a string and two chars
 - the function scans the string and replaces every occurrence of the first char with the second one.
- write a program to test the above function
 - the program should read a string from the user (no spaces) and two characters, then call the function with the input, and print the result.
- example
 - input: "papa", 'p', 'm'
 - output: "mama"

Solution

```
void replace(char str[], char replace_what,  
             char replace_with)  
{  
    int i;  
  
    for (i = 0; str[i] != '\0'; ++i)  
    {  
        if (str[i] == replace_what)  
            str[i] = replace_with;  
    }  
}
```

Solution

```
#define STRING_LEN 100

int main(void)
{
    char str[STRING_LEN + 1];
    char replace_what, replace_with, tmp;

    printf("Please enter a string (no spaces)\n");
    scanf("%100s", str);

    printf("Letter to replace: ");
    scanf(" %c", &replace_what);
    do {tmp=getchar();} while (tmp!='\n');

    printf("Letter to replace with: ");
    scanf(" %c", &replace_with);

    replace(str, replace_what, replace_with);
    printf("The result: %s\n", str);
    return 0;
}
```

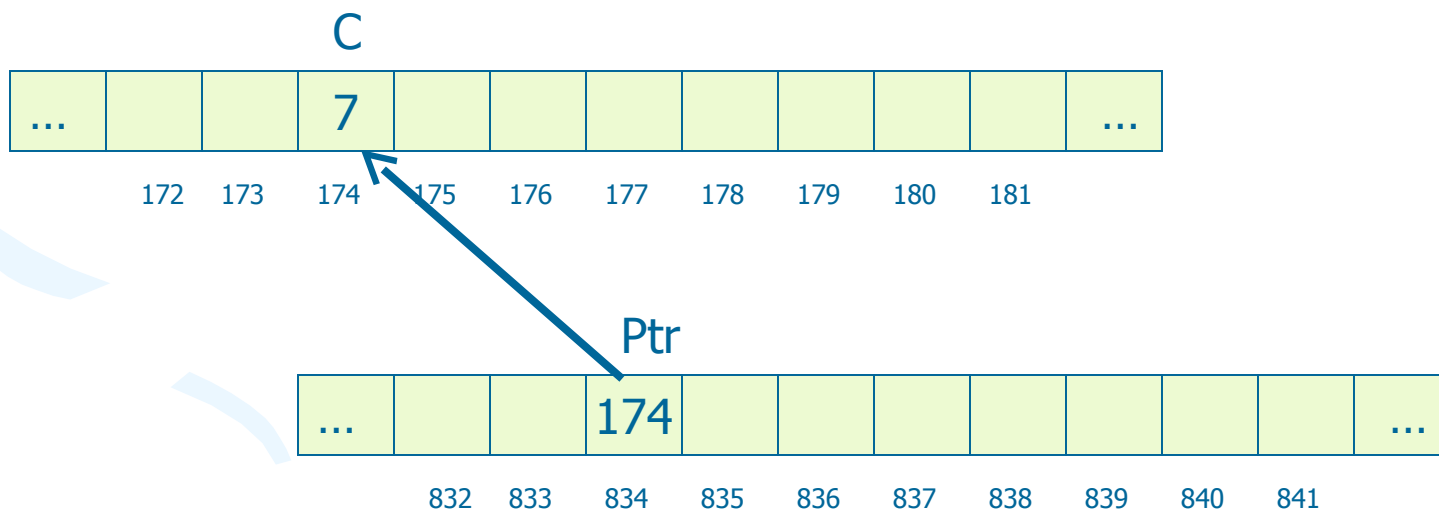
Pointer - Declaration

```
type *variable_name;
```

- A pointer is declared by adding a * before the variable name.
- Pointer is a variable that contains an address in memory.
- The address should be the address of a variable or an array that we defined.

Pointers

- Here ptr is said to *point* to the address of variable c



Referencing and Dereferencing

```
int n;  
int *iptr; /* Declare P as a pointer to int */  
n = 7;  
iptr = &n;  
  
printf("%d", *iptr); /* Prints out '7' */  
*iptr = 177;  
printf("%d", n); /* Prints out '177' */  
iptr = 177; /* This is unadvisable!! */
```

Exercises

Write a function that accepts a double parameter and returns its integer and fraction parts.

Write a program that accepts a number from the user and prints out its integer and fraction parts, using this function.

Solution

```
void split(double num, int *int_part, double *frac_part)
{
    *int_part = (int)num;
    *frac_part = num - *int_part;
}

int main(void)
{
    double num, fraction;
    int integer;

    printf("Please enter a real number: ");
    scanf("%f", &num);

    split(num, &integer, &fraction);
    printf("The integer part is %d\n", integer);
    printf("The remaining fraction is %f\n", fraction);

    return 0;
}
```

Exercise

- Write a function with the prototype:

```
void replace_char(char *str,  
                 char c1,  
                 char c2);
```
- It replaces each appearance of `c1` by `c2` in the string `str`.
Do not use the `[]` operator!
- Demonstrate your function with a program that uses it

Solution

```
void replace_char(char *str, char c1, char c2)
{
    if (str == NULL)
        return;

    while (*str != '\0')
    {
        if (*str == c1)
            *str = c2;

        ++str;
    }
}
```

Command line arguments

- Command line arguments are arguments for the `main` function
 - Recall that `main` is basically a function
 - It can receive arguments like other functions
 - The 'calling function' in this case is the operating system, or another program

'main' prototype

```
int main(int argc, char* argv[])
```

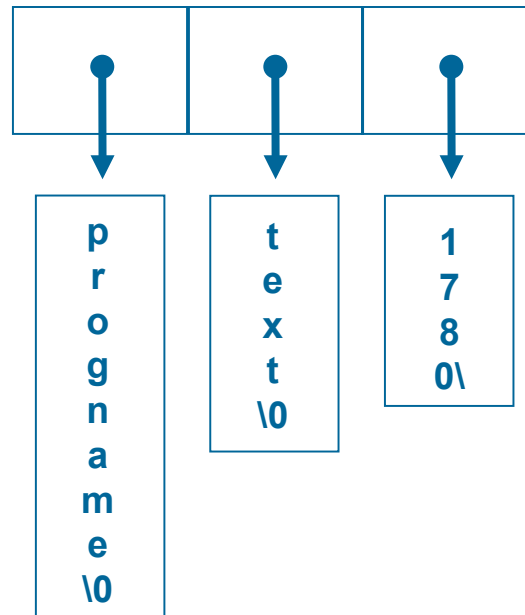
- When we want `main` to accept command line arguments, we must define it like this
 - `argc` holds the number of arguments that were entered by the caller
 - `argv` is an array of pointers to char – an array of strings – holding the text values of the arguments
- The first argument is always the program's name

'main' prototype

```
int main(int argc, char* argv[])
```

argc : 3

argv :



Exercise

- Write a program that accepts two numbers as command line arguments, representing a rectangle's height and width (as floating-point numbers).
- The program should display the rectangle's area and perimeter

Solution

```
int main(int argc, char* argv[])
{
    double width, height;

    if (argc != 3)
    {
        printf("Wrong number of arguments!\n");
        return 1;
    }

    width = atof(argv[1]);
    height = atof(argv[2]);

    printf("The rectangle's area is %f\n", width * height);
    printf("The rectangle's perimeter is %f\n",
           2 * (width + height));

    return 0;
}
```


Homework

- Write a command line program that calculates e^x with the following syntax:
- E 50



File Handling

- **C communicates with files using a new datatype called a file pointer.**
- **File pointer:**
 - references a disk file.
 - used by a stream to conduct the operation of the I/O functions.
- **FILE *fptr;**

A decorative background on the left side of the slide features a light green balloon at the top, a light blue balloon in the middle, and a light purple balloon at the bottom. Yellow streamers and triangular flags are scattered around the balloons.

4 major operations

- Open the file
- Read from a file → program
- Write to a file: Program → file
- Close the file.

Opening a file

- `fopen()` function.
- `FILE *fopen(const char *filename, const char *mode);`

```
FILE *fptr;  
if ((fptr = fopen("test.txt", "r")) ==  
    NULL) {  
    printf("Cannot open test.txt file.\n");  
    exit(1);  
}
```

Opening a file

- filename: name of the file.
 - It can be a string literal: `"data.txt"`
 - It may contain the full path of the file:
`"/root/hedspi/CProgrammingBasic/Lab1/data.txt"`
 - It may be a character array that contains the file name:

```
char file_name[] = "junk.txt";
```
- **NOTE:** *If the file path is not specified, the file is located in the same folder as the C program.*

Mode for text file

mode	Description
"r"	opens an existing text file for reading.
"w"	creates a text file for writing.
"a"	opens an existing text file for appending.
"r+"	opens an existing text file for reading or writing.
"w+"	creates a text file for reading or writing.
"a+"	opens or create an existing text file for appending.

Mode for binary file

mode	Description
"rb"	opens an existing binary file for reading.
"wb"	creates a binary file for writing.
"ab"	opens an existing binary file for appending.
"r+b"	opens an existing binary file for reading or writing.
"w+b"	creates a binary file for reading or writing.
"a+b"	opens or create an existing binary file for appending.

Closing a file

- The `fclose` command can be used to disconnect a file pointer from a file.
- `int fclose(FILE *stream);`

Example: File Open and Close

```
1: /* Opening and closing a file */
2: #include <stdio.h>
3:
4: enum {SUCCESS, FAIL};
5:
6: main(void)
7: {
8:     FILE *fptr;
9:     char filename[] = "haiku.txt";
10:    int reval = SUCCESS;
11:
12:    if ((fptr = fopen(filename, "r")) == NULL){
13:        printf("Cannot open %s.\n", filename);
14:        reval = FAIL;
15:    } else {
16:        printf("The value of fptr: 0x%p\n", fptr);
17:        printf("Ready to close the file.");
18:        fclose(fptr);
19:    }
20:
21:    return reval;
22: }
```

Reading and Writing Disk Files

- In C, you can perform I/O operations in the following ways:
 - **Read or write one character at a time.**
 - **Read or write one line of text (that is, one character line) at a time.**
 - Read or write one block of characters at a time.

Character based file operations in UNIX

- **Read or write one character at a time.**
- Character input and output
 - fgetc() and fputc()
- `int fgetc(FILE *stream);`
- `int fputc(int c , FILE *stream);`

Exercise F1

- Create a text file name lab1.txt with the content as you want.
- Write a program to read from a text file one character at a time, then write it to a new file with the name lab1w.txt

Solution

```
#include <stdio.h>
enum {SUCCESS, FAIL};

void CharReadWrite(FILE *fin, FILE *fout)
{
    int c;
    while ((c=fgetc(fin)) != EOF){
        fputc(c, fout); /* write to a file */
        putchar(c);
        /* display character on the screen */
    }
}
```

Solution

```
enum {SUCCESS, FAIL};
```

```
main(void) {  
    FILE *fptr1, *fptr2;  
    char filename1[] = "lab1a.txt";  
    char filename2[] = "lab1.txt";  
    int reval = SUCCESS;  
  
    if ((fptr1 = fopen(filename1, "w")) == NULL){  
        printf("Cannot open %s.\n", filename1);  
        reval = FAIL;  
    } else if ((fptr2 = fopen(filename2, "r")) == NULL){  
        fclose(fptr1);  
        printf("Cannot open %s.\n", filename2);  
        reval = FAIL;  
    } else {  
        CharReadWrite(fptr2, fptr1);  
        fclose(fptr1);  
        fclose(fptr2);  
    }  
    return reval;  
}
```



Home work

- Write the command `cp` by your self to copy a text file to another
- `mycp a1.txt a2.txt`

Exercise (cont)

- Write a program to read sentences from a specified file one character at a time.
- Each capital letter is converted into a lower-case letter, and each lower-case letter is converted into a capital letter. The new sentence is then written into another file.
- Note that you must output numbers, the signs as they are.

Solution

- Just modify the function CharReadWrite and character manipulate functions in <ctype.h>

```
void CharReadWrite(FILE *fin, FILE *fout)
{
    int c;
    while ((c=fgetc(fin)) != EOF) {
        if (islower(c)) c=toupper(c);
        else if (isupper(c)) c=tolower(c);
        fputc(c, fout); /* write to a file */
        putchar(c);
        /* display character on the screen */
    }
}
```

Read or write one line at a time.

- Two functions: `fgets()` and `fputs()`
- `char *fgets(char *s, int n, FILE *stream);`
 - `s` references an array that is used to store characters
 - `n` specifies the maximum number of array elements.
- `fgets()` function can read up to `n-1` characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered.

Read or write one line at a time.

- `int fputs(const char *s, FILE *stream);`
- `s`: array that contains the characters to be written to a file
- return value
 - 0 for success
 - non zero in case of fail.

Exercise

- Redo the exercise F1 but the program will read and write one character line at a time.

Solution

```
#include <stdio.h>
enum {SUCCESS, FAIL, MAX_LEN = 81 };

void LineReadWrite(FILE *fin, FILE
    *fout)
{
    char buff[MAX_LEN];
    while (fgets(buff, MAX_LEN, fin) !=
        NULL) {
        fputs(buff, fout);
        printf("%s", buff);
    }
}
```

Solution

```
main(void) {
    FILE *fptr1, *fptr2;
    char filename1[] = "lab1a.txt";
    char filename2[] = "lab1.txt";
    int reval = SUCCESS;

    if ((fptr1 = fopen(filename1, "w")) == NULL) {
        printf("Cannot open %s.\n", filename1);
        reval = FAIL;
    } else if ((fptr2 = fopen(filename2, "r")) == NULL) {
        printf("Cannot open %s.\n", filename2);
        reval = FAIL;
    } else {
        LineReadWrite(fptr2, fptr1);
        fclose(fptr1);
        fclose(fptr2);
    }
    return reval;
}
```

Exercise

- Write a program named mycat that read and display on the screen the content of a given file. The command can take 1 or 2 arguments
- `cat <filename>` : display content to the end
- `cat <filename> -p` : view page by page.

Read and write formatted text

- `int fscanf(FILE *stream, const char *format, ...);`
 - This function works like `scanf` except that it read from a file stream.
- `int fprintf(FILE *stream, const char *format, ...);`
 - The only difference between `fprintf` and `printf` is that `fprintf` can redirect output to a particular stream.

Homework

- Write a program to read a text file created with emacs. Put a line number to the head of the line and output the contents of the file to the standard output. A text file name can be specified as the argument to the program.
- For example, the following content of a text file
This is sample file.
Hello!
- is output as follows.
1 This is sample file.
2 Hello!

The slide features a decorative background on the left side with a light green balloon at the top, a light blue balloon in the middle, and a light purple balloon at the bottom. Yellow streamers and triangular flags are scattered around the balloons. The main content is centered on the right side of the slide.

Home work

Write a program to compare two files given as the command parameters and indicates:

- the first line where they differ(line numbers).
- all lines where they differ.

See you next time

