HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

C Basic

Basic data types, struct, memory allocation, binary file

ONE LOVE. ONE FUTURE.

Content

- Struct
- Dynamic memory allocation
- Binary file
- Exercises

HUST

hust.edu.vn fb.com/dhbkhn

Dynamic memory allocation

Dynamic memory allocation

- Regular arrays have a fixed size used to store a maximum number of elements known at compile time. This size cannot be changed after the program is created. However, we can't always know in advance how many elements the program will need to work with.

- Dynamic memory allocation technique
  - Request the system to allocate memory at runtime
  - The allocated memory is managed by a pointer

Hàm `malloc`

void * malloc(unsigned int nbytes);

- Request the system to allocate a block of memory of size nBytes.
- malloc returns a pointer to the allocated memory if the allocation is successful, and returns a NULL pointer if it fails.
- Note: Always check if the memory was allocated successfully or not.
- Part of the stdlib.h library: #include <stdlib.h>

Example

```
#include <stdlib.h>
int main() {
   int i, n, * p;
   printf("How many numbers do you want to enter?\n");
   scanf("%d", &n);
   p = (int*)malloc(n * sizeof(int));
   if (p == NULL) {
      printf("Memory allocation failed!\n");
      return 1;
   }
   /* input integer numbers*/
   ...
      /* print reverse way*/
      ...
      free(p); /* free mem*/
   return 0;
}
```

Hướng dẫn

```
int main()
{
   . . .
   /* get the numbers – dynamic memory allocated aray */
   printf("Please enter numbers now:\n");
   for (i = 0; i < n; i++)
      scanf("%d", &p[i]);

   /* print in reverse way*/
   printf("The numbers in reverse order are - \n");
   for (i = n - 1; i >= 0; --i)
      printf("%d ", p[i]);
   printf("\n");
   free(p);
   return 0;
}
```

Type cast ?

Type casting in the statement

```
p = (int *)malloc(nsizeof(int));
```

is necessary because the malloc function returns void *:

```
void * malloc(unsigned int nbytes);
```

The (void *) type specifies a general pointer that can be cast to any pointer type.

Function calloc

void *calloc(size_t nitems, size_t size);

- Dynamic memory allocation involves a specified number of items, nitems, all of the same type, each with a size of size bytes.
- The initialization of these elements with a default value of 0 isn't performed by the malloc function. It returns a pointer to the allocated memory if the allocation request is successful, and it returns a NULL pointer if it fails.
- **ptr = (float*) calloc(25,sizeof(float));**

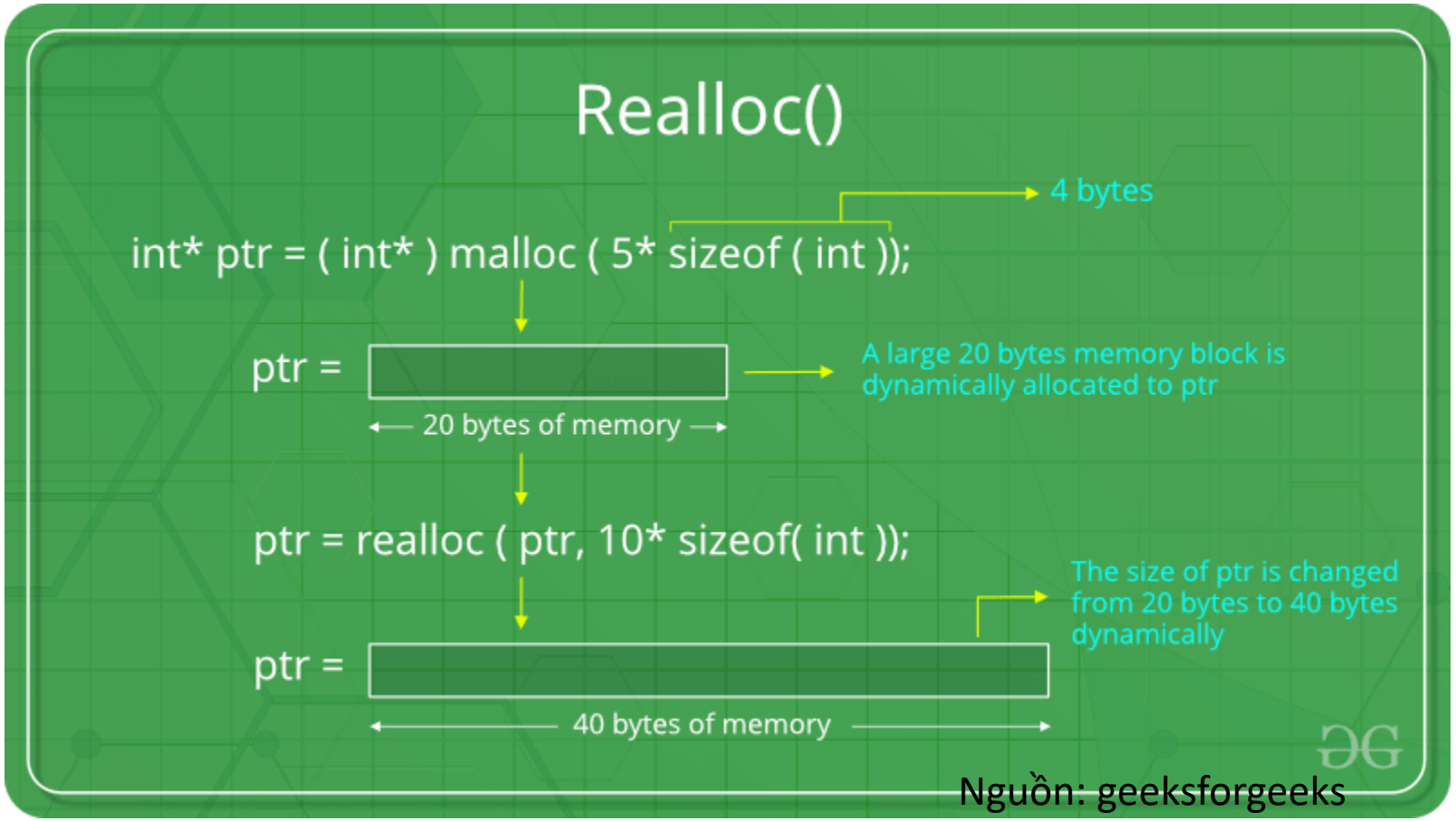
Function Reallocate

- Sometimes, a program needs to allocate additional memory after the initial allocation.

```
void *realloc(void *ptr, size_t size)
```

- The realloc function is used to change the size of the memory block pointed to by the pointer ptr, which was previously allocated by malloc or calloc.
- Parameters:
  - ptr: A pointer to the memory block that needs to be reallocated. If this pointer is NULL, a new memory block is allocated and returned by the function.
  - size: The new size of the memory block in bytes. If it's 0 and ptr is pointing to a previously allocated memory block, the memory block pointed to by ptr is deallocated, and the function returns NULL.
- Return Value:
  - It returns a pointer to the memory block if the reallocation request is successful, and it returns a NULL pointer if it fails.

Function Reallocate



Nguồn: geeksforgeeks

Example

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
   char* str;
   str = (char*)malloc(15);
   strcpy(str, "tutorialspoint");
   printf("String = %s,  Address = %u\n", str, str);
   /* realloc*/
   str = (char*)realloc(str, 25);
   strcat(str, ".com");
   printf("String = %s,  Address = %u\n", str, str);
   free(str);
   return(0);
}
```

Free memory

```
void free(void *ptr);
```

- The free(p) function releases the memory block pointed to by p.
- If p doesn’t point to any memory block, a program execution error will occur.
- Always remember to deallocate dynamically allocated memory when it's no longer needed, for instance, when exiting the program.

Example

- Example 1.
- Implementing the my_strcat function:
  - Input: two strings, s1 and s2
  - Output: a pointer to dynamically allocated memory containing the concatenated string of s1 and s2
- Example: Concatenating "hello_" and "world!" results in "hello_world!"

- Utilize dynamic memory allocation techniques
- Perform checks to validate the implemented functions

Solution - my_strcat

```
char* my_strcat(char* str1, char* str2) {
    int len1, len2;
    char* result;
    len1 = strlen(str1);
    len2 = strlen(str2);
    result = (char*)malloc((len1 + len2 + 1) * sizeof(char));
    if (result == NULL) {
        printf("Allocation failed! Check memory\n");
        return NULL;
    }
    strcpy(result, str1);
    strcpy(result + len1, str2);
    return result;
}
```

Solution - my_strcat

```
int main() {
    char str1[MAX_LEN + 1], str2[MAX_LEN + 1];
    char* cat_str;
    printf("Please enter two strings\n");
    scanf("%100s", str1);
    scanf("%100s", str2);
    cat_str = my_strcat(str1, str2);
    if (cat_str == NULL) {
        printf("Problem allocating memory!n");
        return 1;
    }
    printf("The concatenation of %s and %s is %s\n", str1, str2, cat_str);
    free(cat_str);
    return 0;
}
```

Exercise

- Exercise 1. Build and execute the demonstration (demo) of the following function:
  - char* subStr(char* s1, int offset, int number)

- This function extracts a substring from string s1, starting from the character at index offset (counting from 0) and of length number.

- Ensure to validate the arguments' validity. In case the value of number exceeds the remaining length of string s1 from position offset, return the remaining part of s1 from position offset.

Struct

Struct

- Combining multiple variables into a complex entity under a single name is known as structuring.
- It's a convenient way to group related information, all pertaining to the same entity within a problem. The variables collectively contribute to creating a structure known as components or fields within a struct.

```
struct struct-name
{
    field-type1 field-name1;
    field-type2 field-name2;
    field-type3 field-name3;
       ...
};
```

Example

Example : define struct of complex number

```
struct complex {
    int real;
    int img;
};
struct complex num1, num2, num3;
```

Using typedef

```
typedef struct complex {
    int real;
    int img;
} complex_t;


complex_t num1, num2;
```

Example

Example 1. Given two structs:

- Write a function named is_in_circle that returns 1 if the point p is inside the circle c.
- Test the function using a program.

```
typedef struct point
{
   double x;
   double y;
} point_t;

typedef struct circle
{
   point_t center;
   double  radius;
} circle_t;
```

Solution

```
int is_in_circle(point_t* p, circle_t* c) {
   double x_dist, y_dist;
   x_dist = p->x - c->center.x;
   y_dist = p->y - c->center.y;
   return (x_dist * x_dist + y_dist * y_dist <= c->radius * c->radius);
}
int main() {
   point_t  p;
   circle_t c;
   printf("Enter point coordinates\n"); scanf("%lf%lf", &p.x, &p.y);
   printf("Enter circle center coordinates\n");
   scanf("%lf%lf", &c.center.x, &c.center.y);
   printf("Enter circle radius\n"); scanf("%lf", &c.radius);
   if (is_in_circle(&p, &c))
      printf("point is in circle\n");
   else
      printf("point is out of circle\n");
   return 0;
}
```

Exercise

- Exercise 1. Write a function to check if two circles intersect. Develop a program that creates a dynamic array of elements, where each element represents a circle. The number of elements should be provided by the user at runtime.

- The program should allow users to:
  - Manually input information for each circle
  - Automatically generate random information for each circle
- Utilize the function mentioned earlier and display:
  - Information about the circles
  - Information about the circle(s) that intersect(s) with the most number of other circles (along with detailed information, such as which circles it intersects with).

Exercise

- Exercise 2. Enhance last week's assignment about the student list and grades table as follows:

- Instead of using a regular array, the program should dynamically allocate the exact memory needed to store the data read from the class list file.

- Add an additional feature for additional input: the program should prompt for the number of students to be added, then dynamically reallocate memory to accommodate new data using the realloc function.

- Note: Read a class list file with at least 10 lines, and ensure that the data accurately represents reality

HUST

hust.edu.vn fb.com/dhbkhn

Working with binary file

Binary modes

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.

C in C provides two I/O functions: fread() and fwrite(), allowing operations of input and output based on blocks of bytes. Similar to other functions, these functions work with arguments that are file pointers.

fread()

- Function fread

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

- ptr is a pointer pointing to an array used to store data read from the file.
- size: size of each array element (in bytes).
- n: the number of data elements read from the file.
- stream: a file pointer associated with the file being read or written.
- The function returns the number of elements successfully read from the file.

fwrite()

- Function fwrite

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);
```

- ptr is a pointer pointing to an array used to store data write to the file.
- size: size of each array element (in bytes).
- n: the number of data elements write to the file.
- stream: a file pointer associated with the file being read or written.
- The function returns the number of elements successfully write to the file.

Function feof

```
int feof(FILE *stream);
```

- Check if the file pointer is at the end of the file or not.
- The function returns 0 if the pointer has not reached the end of the file; otherwise, it returns a non-zero value.

Example

- Example 1. Read 80 byte from file haiku.txt.

```
#define MAX_LEN  80
int num;
FILE* fptr2;
char filename2[] = "haiku.txt";
char buff[MAX_LEN + 1];
if ((fptr2 = fopen(filename2, "r")) == NULL) {
    printf("Cannot open %s.\n", filename2);
    reval = FAIL; exit(1);
}
. . . .
num = fread(buff, sizeof(char), MAX_LEN, fptr2);
buff[num * sizeof(char)] = '\0';
printf("%s", buff);
```

Example

- Example 2. Write a program to copy a file (from lab1.txt to lab1a.txt) using block-level file operations similar to previous exercises involving file reading and writing
- Using functions: fread, fwrite, feof

```
#include <stdio.h>
enum { SUCCESS, FAIL };
#define MAX_LEN  80

void BlockReadWrite(FILE* fin, FILE* fout) {
   int num;
   char buff[MAX_LEN + 1];

   while (!feof(fin)) {
      num = fread(buff, sizeof(char), MAX_LEN, fin);
      buff[num * sizeof(char)] = `\0';

         printf("%s", buff);
      fwrite(buff, sizeof(char), num, fout);
   }
}
```

Solution

```
int main() {
   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 {
      BlocReadWrite(fptr2, fptr1);
      fclose(fptr1);
      fclose(fptr2);
   }
   return reval;
}
```

Example

- Example 3. Write a program named 'mycat' that functions similarly to the 'cat' command in Unix using block-level file operations.
- Suggestions:
  - Accept command-line arguments.
  - Utilize the 'fread' function.

```
void BlockCat(FILE* fin) {
   int num;
   char buff[MAX_LEN + 1];

   while (!feof(fin)) {
      num = fread(buff, sizeof(char), MAX_LEN, fin);
      buff[num * sizeof(char)] = `\0';

         printf("%s", buff);
   }
}
```

Example

```
main(int argc, char* argv[]) {
   FILE* fptr1, * fptr2;
   int reval = SUCCESS;
   if (argc != 2) {
      printf("The correct syntax should be: cat1 filename \n");
      reval = FAIL;
   }
   if ((fptr1 = fopen(argv[1], "r")) == NULL) {
      printf("Cannot open %s.\n", argv[1]);
      reval = FAIL;
   }
   else {
      BlockCat(fptr1);
      fclose(fptr1);
   }
   return reval;
}
```

Exercise

- Exercise 1. Write a program that copies files in various modes, operating with a command-line menu interface with the following main functions:
  - Copy by character
  - Copy by line
  - Copy by block - optional size
  - Quit
- For each copy function, after completing the copying process, display the execution time in milliseconds for comparison.

- Note: The source file must be a text file with a minimum size of 640KB

Exercise

- Exercise 2. Read and write a binary file containing structures.
- Assume you need to manage your phone book using a program. Define a structure representing a phone book with fields such as 'name,' 'telephone number,' 'e-mail address,' and declare an array capable of holding a maximum of 100 elements of this structure type.
- Input data for around 10 elements of the array.
- The program should then write the contents of the array with the aforementioned elements to a file named 'phonebook.dat' using the 'fwrite' function.
- Read the data from the file back into the array using the 'fread' function and print the content of the array to the screen to verify

Solution

```
enum { SUCCESS, FAIL };
#define MAX_ELEMENT  20

// the phone book structure
typedef struct phoneaddress_t {
   char name[20];
   char tel[11];
   char email[25];
}phoneaddress;

int main(void)
{
   FILE* fp;
   phoneaddress phonearr[MAX_ELEMENT];
   int i, n, irc; // return code
   int reval = SUCCESS;
```

Solution

```
printf("How many contacts do you want to enter (<20)?");
scanf("%d", &n);
for (i = 0; i < n; i++) {
    printf("name:"); scanf("%s", phonearr[i].name);
    printf("tel:"); scanf("%s", phonearr[i].tel);
    printf("email:"); scanf("%s", phonearr[i].email);
}
if ((fp = fopen("phonebook.dat", "w+b")) == NULL) {
    printf("Can not open %s.\n", "phonebook.dat");
    reval = FAIL;
}
// write the entire array into the file
irc = fwrite(phonearr, sizeof(phoneaddress), n, fp);
printf(" fwrite return code = %d\n", irc);
fclose(fp);
```

Solution

```
    //read from this file to array again
    if ((fp = fopen("phonebook.dat", "rb")) == NULL) {
        printf("Can not open %s.\n", "phonebook.dat");
        reval = FAIL;
    }
    irc = fread(phonearr, sizeof(phoneaddress), n, fp);
    printf(" fread return code = %d\n", irc);
    for (i = 0; i < n; i++) {
        printf("%s-", phonearr[i].name);
        printf("%s-", phonearr[i].tel);
        printf("%s\n", phonearr[i].email);
    }
    fclose(fp);
    return reval;
}
```

Exercise

- Exercise 3. Write a program that reads the grade file ('bangdiem.txt' - the result of the exercise on the student list and given grades) and stores the data using dynamic memory allocation, then writes it to a binary file 'grade.dat' (containing an array of structures about students).
- The program should: read the 'grade.dat' file and display the grade table on the screen, search for a student based on their student ID, update the grades with new input from the user, and save them to the file.
- The program should be designed with an interactive command-line menu:
- 1. TextToDat
- 2. Display Dat file
- 3. Search and Update
- 4. Quit.

HUST

hust.edu.vn fb.com/dhbkhn

Access binary file randomly

Access binary file randomly

- Using two functions: fseek() and ftell()
- fseek(): This function moves the file pointer to a desired location within the file.
- fseek(FILE *stream, long offset, int whence);
- Stream: File pointer
- Offset: Length of translation in bytes.
- Whence: Constant indicating the reference point and direction of translation
  - SEEK_SET: From the beginning of the file, shift towards the end of the file.
  - SEEK_CUR: From the current file pointer position, shift towards the end of the file.
  - SEEK_END: From the end of the file, shift towards the beginning of the file

Access binary file randomly

- Function ftell: Provides the current position value of the file pointer.
- Syntax: long ftell(FILE *stream);

- Function rewind(): Sets the file pointer position back to the beginning of the file.
- Syntax: void rewind(FILE *stream);

Example : Read binary file

- Example 1. Write a program that reads a specific portion of the contact data stored in the file 'phonebook.dat.' For example, data from the 2nd to the 3rd entry, or from the 3rd to the 6th. Then, modify the value of the email field and rewrite it to the file at the exact extracted positions.

- The program needs to allocate memory to store the correct amount of data read from the file when running the program. For instance, you might require a dynamic array with 4 structure elements to store data from the 3rd to the 6th entry.

Solution

```
#include <stdio.h>
#include <stdlib.h>
enum { SUCCESS, FAIL };
#define MAX_ELEMENT  20
// the phone book structure
typedef struct phoneaddress {
    char name[20];
    char tel[11];
    char email[25];
}phoneaddress;
int main(void) {
    FILE* fp;
    phoneaddress* phonearr;

    int i, n, irc; // return code
    int reval = SUCCESS;
    printf("Read from 2sd data to 3rd data \n");
```

Solution

```
if ((fp = fopen("phonebook.dat", "r+b")) == NULL) {
    printf("Can not open %s.\n", "phonebook.dat");
    reval = FAIL;
}
// Memory allocation
phonearr =
    (phoneaddress*)malloc(2 * sizeof(phoneaddress));
if (phonearr == NULL) {
    printf("Memory allocation failed!\n");
    return FAIL;
}
if (fseek(fp, 1 * sizeof(phoneaddress), SEEK_SET) != 0)
{
    printf("Fseek failed!\n");
    return FAIL;
}
irc = fread(phonearr, sizeof(phoneaddress), 2, fp);
```

Solution

```
    for (i = 0; i < 2; i++) {
        printf("%s-", phonearr[i].name);
        printf("%s-", phonearr[i].tel);
        printf("%s\n", phonearr[i].email);
    }
    // Modifying some data
    strcpy(phonearr[1].name, "Lan Hoa");
    strcpy(phonearr[1].tel, "0923456");
    strcpy(phonearr[1].email, "lovelybuffalo@hut.edu.vn");
    fseek(fp, 1 * sizeof(phoneaddress), SEEK_SET);
    irc = fwrite(phonearr, sizeof(phoneaddress), 2, fp);
    printf(" fwrite return code = %d\n", irc);
    fclose(fp); free(phonearr);
    return reval;
}
```

Exercise

- Exercise 1. Write a program to convert data from a Vietnamese-English dictionary from text format to binary.
- The data is available at: http://www.denisowski.org/Vietnamese/vnedict.txt
  - Miên Đất Hứa : the Promised Land
  - Miến : Burma (short for Miến Điện)
  - Miến Điện : Burma
  - Miền Trung : Central Vietnam
- Afterward, the program should read the data from the binary file, prompt the user for the starting and ending positions of the entries, and display the words located at these positions in the dictionary.

Exercise : Phone information management

- Exercise 2. Managing mobile phone specifications
- From the websites of mobile phone showrooms, create a text file named PhoneDB.txt containing information about at least 20 recent phone models such as iPhone, Samsung, Oppo, Huawei, each on a separate line, following this format:
- Model Memory Space (GB) Screen Size (inches) Price
- Write a program with the following menu interface:
  - Import DB from text: Read the file PhoneDB.txt and convert it into the binary format PhoneDB.dat using dynamic memory allocation techniques.
  - Import from DB: Read data from the file PhoneDB.dat and load it into the program's memory. Allow users to choose between two read modes: reading the entire database and reading a portion (specify the start and end positions of the record).
  - Print All Database: Display information about the phone models on the screen, with each model on a separate line and columns aligned.
  - Search by phone by Phone Model: Search for a phone based on the model entered by the user.
  - Exit

Exercise : Split and merge File

- Exercise 3. Split and merge File

  Using the phonebook.dat file (result of the previous Lab exercise) containing at least 20 contact numbers, write the following programs:

  - The 'filesplit' program takes two arguments: the source file name (.dat) and an integer N, and the names of two result files. It will split the source file into two files. The first file contains the first N contact numbers, and the second file contains the remaining contact numbers. For example:
  - filesplit phone.dat 10 phone1.dat phone2.dat
  - The 'filemerge' program merges two previously split files into one file:
- filemerge phone1.dat phone2.dat phone.dat
- The 'fileread' program reads and displays the list of contact numbers contained in any .dat file on the screen. It is used to verify the results of executing the 'filesplit' and 'filemerge' programs.