Problem-solving using arrays

Arrays in problem solving

USE OF ARRAYS: NON-ORDERED SEQUENCES, ORDERED SEQUENCES, INDEX-VALUE CORRESPONDENCE

Arrays in problem solving

- Arrays are a collection of data of the same type

- Containers of homogeneous values
  - Without ordering criteria
  - With ordering criteria
  - Leveraging index-value correspondence

Arrays as non-ordered collection of values

- Array is a simple collection of data.
- Order does not count
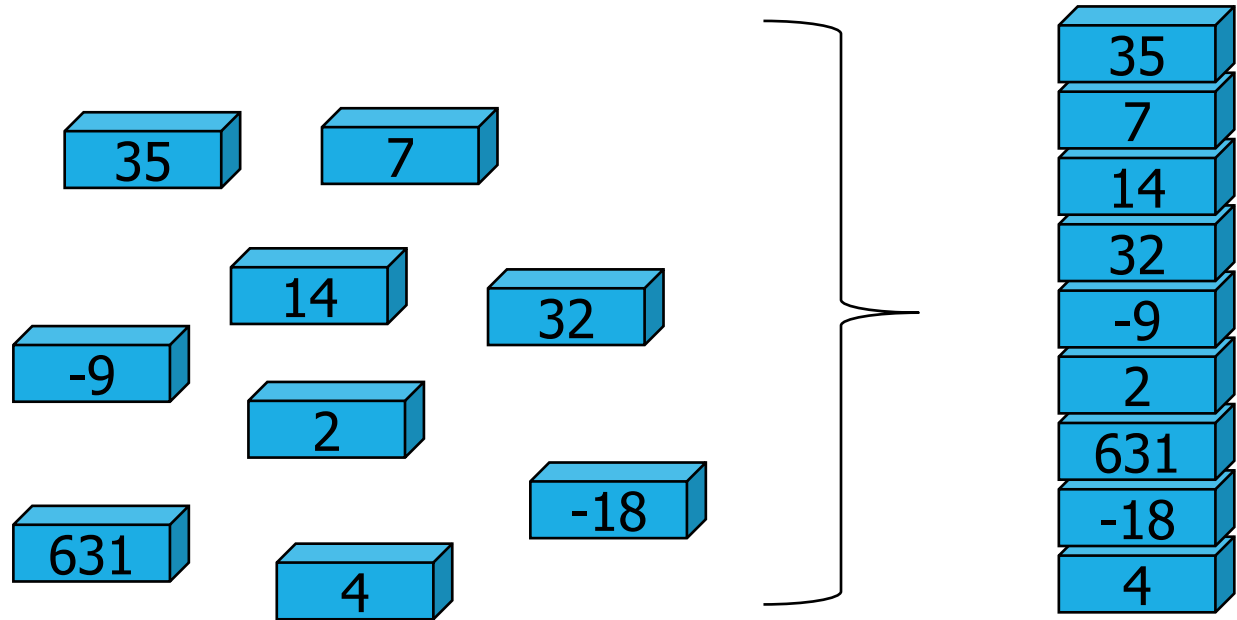
# Arrays as non-ordered collection of values

- When:
  - o Problems where we need to collect multiple values, without any order relation
  - o Each value can be stored in any position of the array, as the position is not important for the solution of the problem
  - o Values can be accessed by simple iteration on all the indexes

- Exmples:
  - Collecting input data for future processing, or prepare data for output
  - Same operation to multiple values

Ordered arrays

- Values in the array are organized with some ordering criteria (ex. Increasing values)

35   7

14   32

-9

2

631   -18

4

-18
-9
2
4
7
14
32
35
631
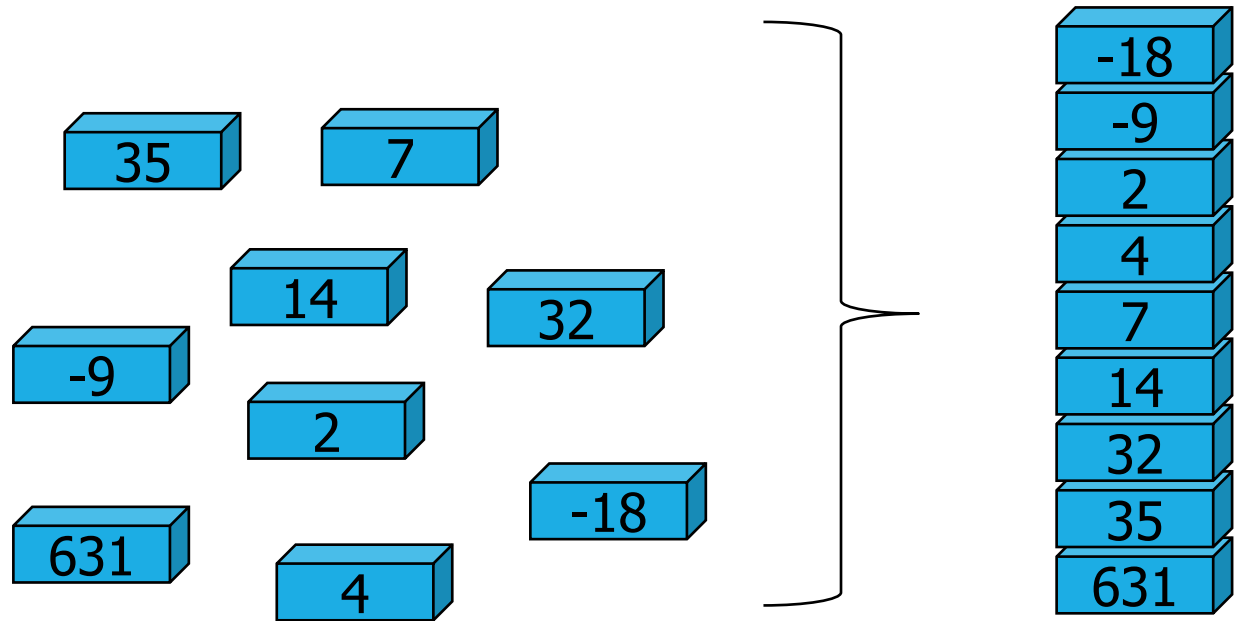
Ordered arrays

When:

- o Problems when we need to create/maintain an ordering criterium for a collection of data
- o Most frequently, a mono-dimensional vector: linear ordering (either total or partial)
- o In this case, the index indicates the position in the ordering

Examples:

- Order data with chronologic criteria data that are in the first positions have been acquired first, or vice versa
- Order data according to values (increasing/decreasing)
- Sequences of numerical samples that correspond to physical measurements
- Mathematical / statistical operations on numerical series

Arrays with index ↔ value correspondence

Each index (int in the range 0..NDATA-1) corresponds to a specific value (and vice versa):

0	
1	"January"
2	"February"
3	"March"
4	"April"
5	"May"
6	"June"
7	"July"
8	"August"
…	…

Arrays with index ↔ value correspondence

Each index (int in the range 0..NDATA-1) corresponds to a specific value (and vice versa):

indexes

0
1 "January"
2 "February"
3 "March"
4 "April"
5 "May"
6 "June"
7 "July"
8 "August"
... ...
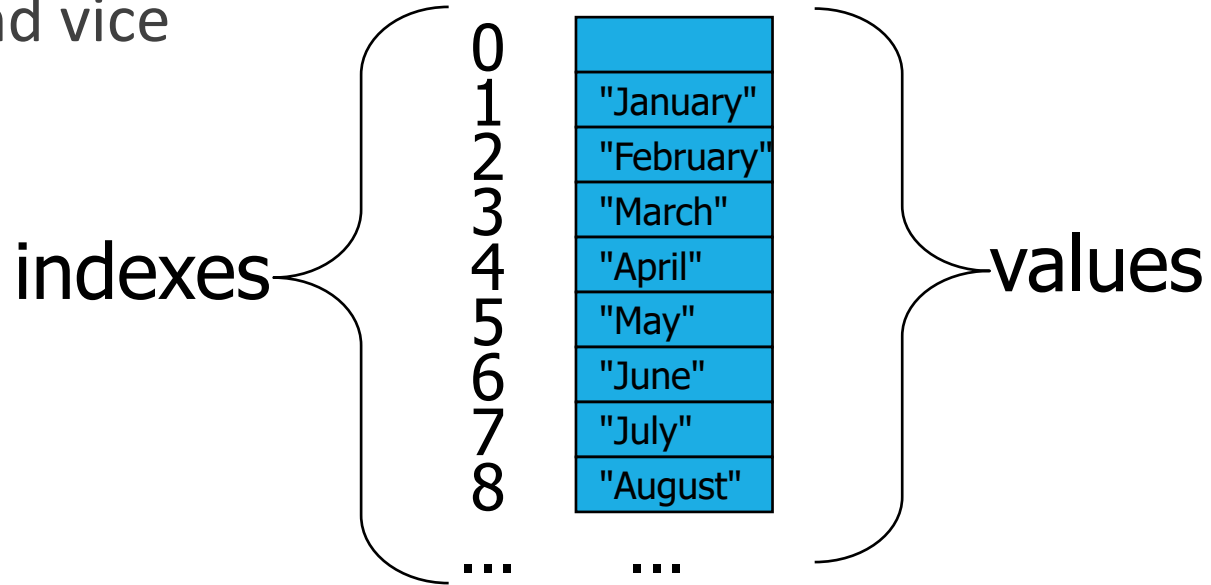
values

Indexes and values

Relation index ↔ value:

- each element of an array is characterized by one (or more, in case of matrices) indexes and one value

- we can take advantage of this organization to establish a meaningful relation between index and value

From index to value

Starting from the index, compute the value. Thanks to the indexing of arrays, this operation is immediate:

6 →

0	
1	"January"
2	"February"
3	"March"
4	"April"
5	"May"
6	"June"
7	"July"
8	"August"
...	...

→ **"June"**
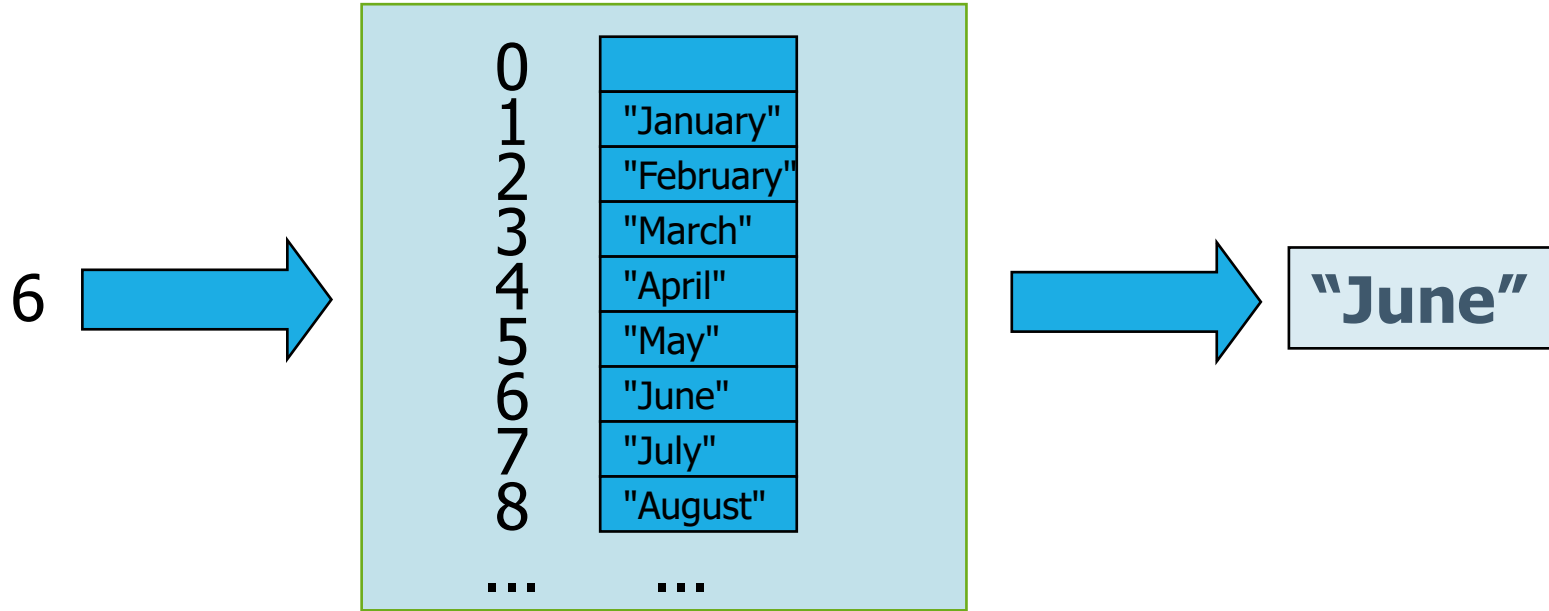
From value to index

Starting from the value, compute the index. This operation is not immediate → we need to search for the value:

0	
1	"January"
2	"February"
3	"March"
4	"April"
5	"May"
6	"June"
7	"July"
8	"August"
...	...

6 ← ... ← **"June"**
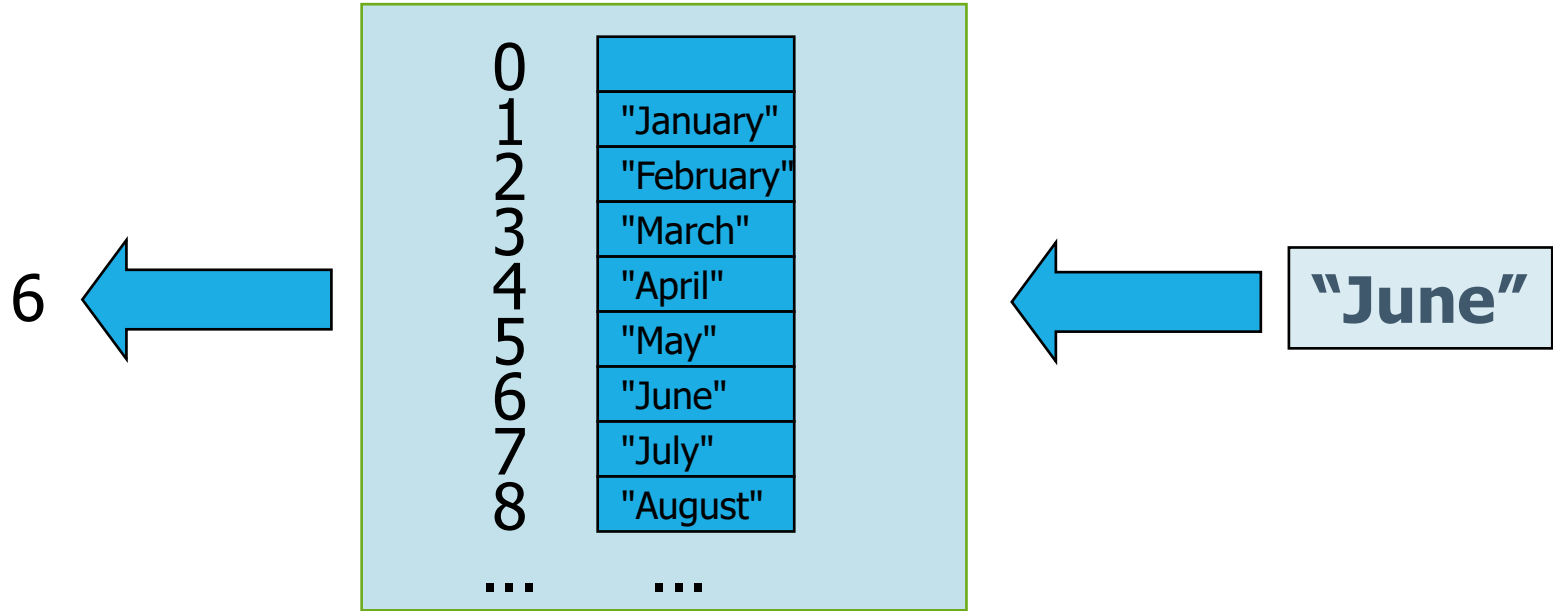
Arrays with index ↔ value correspondence

- When:
  - any problems where there is a meaningful relation between integer numbers and data (either numerical or not)
  - the integer number will be associated to the index of an element, the value to its content

  Careful!
  - The index cannot be too large
  - There may be empty (or null) elements, corresponding to the indexes that are not used

Arrays with index ⟷ value correspondence

Examples:

- Numerical problems (statistics and computations with integer values, computation of functions in the cartesian plane y=f(x), where the x value is integer)
- Numerical encoding/decoding of non-numerical information
- Text processing problems (matrix of characters, corresponding to a page of a text)
- Verification problems (logical flags corresponding to integer values)
- Selection problem (search index corresponding to a given key).

Numerical problems

NUMERICAL PROBLEMS REQUIRING ARRAYS

Numerical problems

- Algebra, geometry, statistics problems, etc., similar to the ones using scalar data

- Arrays can be used:
  o To collect and process (collections of) numbers
  o To represent data with an inherent linear (vectors) or tabular (matrices) structure
  o To handle meaningful relations between numbers (index and value).

Problems on collections of numbers

- Problems where we handle groups of numerical data, with I/O operations, unions, intersections, etc. … in any order. THE ORDER DOES NOT COUNT

- Solution often based on loops or nested loops, to:
  - o access all the elements of the array
  - o access, per each element of the first array, all the elements of a second array

Intersection of sets of numbers

Formulation:

o Acquire from keyboard a first sequence of 10 integers (without repeated elements)

o Acquire from keyboard a second sequence of 10 integers (without repeated elements)

o Compute and print on the screen all the numbers that are shared by the two sequences

Solution:

o It is a problem of intersection of sets. Per each element of the first set, we need to detemine whether this value belongs to the other set, too.

Intersection of sets of numbers

Algorithm:

- o input: two loops to acquire the two sets of values
- o processing: nested loop to compare each element of the first set with each element of the second set (or vice versa)
- o output: loop on the intersection data

The thee steps can be either distinct or (partially) embedded into one another: while we acquire the data, we can compute the intersection and write the ouput

Intersection of sets of numbers

<u>Data structure:</u>

It depends on the algorithm we decide to adopt:

- o We need at least one array for the first set of values
- o The need for a second array depends on which scheme we choose
  - Three distinct steps: input, processing, output → two arrays
  - Optionally, processing and output could be embedded into the input of the second sequence (per each element of the second set of values, we can directly determine whether it is an element of the first array, and print the output accordingly) → one array
- o The following solution adopts the first scheme (you can solve the second by homework). The result of the intersection is stored into the first array.

Solution

```c
#define NDATA 10
#include <stdio.h>
void readVector(int data[], int n);
void printVector(int data[], int n);

int main(void)  i{
  int data0[NDATA], data1[NDATA];
  int i, j, ni, found;
  /* input */
  readVector(data0,NDATA);
  readVector(data1,NDATA);


  /* compute intersection*/
```

```c
  for (i=ni=0; i<NDATA; i++) {
    found=0;
    for (j=0; j<NDATA && (!found); j++) {
      if (data0[i]==data1[j])
        found=1;
    }
    /* if the value is found, store it in the
    first portion of the array data0
    (the one that was already processed) */
    if (found)
      data0[ni++]=data0[i];
  }
  /* output */
  printVector(data0,ni);
}
```

Solution

```c
void readVector (int data[], int n) {
  int i;
  printf("Provide %d integer values (separated by either space or return):\n", n);
  for (i=0; i<n; i++) {
    scanf("%d", &data[i]);
  }
}
void printVector (int data[], int n);
 int i;
  printf("The values are: %d\n", n);
  for (i=0; i<n; i++) {
    printf("%d ", data[i]);
  }
  printf("\n");
}
```

Problems on ordered sequences of data

- Problems with ordered sequences of data, that need to be stored into an array before processing because:
  - it is necessary to wait until the final value before we can start the processing
  - we need to perform repeated operations on all the data

- We cannot handle infinite sequences, but only ordered collections of values, organized into mono- or multi-dimensional arrays (matrices)

Data normalization

Formulation: write a C function that:

- Acquires from a text file a sequence of real numbers separated by either spaces or returns:
  - The number N of values is not a priori known, but it can be over-estimated (for example, maximum value is 100)

- Determines, for each i-th value d_i ($0 \leq i < N$), the average value of all the precedent (p_i) and subsequent (s_i) values

- Write a new sequence of values in a second text file, where the d_i values are normalized as follows:
  - Each value (d_i) are replaced by the arithmetic mean of d_i, p_i and s_i

- The names of the two files are given as argument.

Data normalization

Solution:

o we need to iteratively compute the average values. Then, with a following loop, we need to perform the normalization

Data structure:

o An array to store the data from the input file

o An array to store the average values (? … to be decided)

NB: You may avoid using arrays if you read the file multiple times

o This solution is not suggested: it is better to **avoid reading a file more than one time**!

Data normalization

Algorithm 1: O(N^2)
- Input: array, with values iteratively read from input text
- Processing:
- For each value
  - Compute the mean value of the precedent and mean value of subsequent values (function avg)
  - Compute the mean value of the means, stored in another array (the original one needs to be maintained unchanged, to compute the mean values)
  - The precedent of the first value and the subsequent of the last value do not exist! Two possible strategies:
    - Use a conventional value, for example 0 (adopted strategy)
    - Use the same value of the first/last element
    - Do not consider the non-existing value (compute the mean of only two values)
- Output of the re-computed array.

Data normalization

Example:

Suppose to receive the following six values:

- 4.0 5.0 3.0 2.0 1.0 7.0

Result will be:

- data[0] = (4.0 + 0.0 + (18.0)/5)/3 = 2.53
- data[1] = (5.0 + 4.0/1 +(13.0)/4)/3 = 4.08
- data[2] = (3.0 + 9.0/2 +(10.0)/3)/3 = 3.61
- data[3] = (2.0 + 12.0/3 +(8.0)/2)/3 = 3.33
- data[4] = (1.0 + 14.0/4 +(7.0)/1)/3 = 3.83
- data[5] = (7.0 + 15.0/5 + 0.0)/3 = 3.33

Solution - version 1: O(N^2)

```
#define NMAX 100
float avg(float d[], int i0, int i1);
int readFile(char fileName[], float d[], int nmax);
void writeFile(char fileName[], float d[], int n);
void normalizeNum(char nfin[],char nfout[]){
  float data[NMAX], dataNew[NMAX];
  int i, j, N;
  N = readFile(nfin,data,NMAX); /* input */
  for (i=0; i<N; i++) {
    float pred = avg(data,0,i-1);
    float succ = avg(data,i+1,N-1);
    dataNew[i] = (pred+data[i]+succ)/3;
  }
  writeFile(nfout,dataNew,N); /* output */
}
```

```
float avg(float d[], int i0, int i1) {
  int i;
  float sum;
  if (i0>i1) {
    return 0.0; // no data: assume 0
  }
  sum = 0.0;
  for (i=i0; i<=i1; i++) {
    sum = sum + d[i];
  }
  return sum/(i1-i0+1);
}
```

Solution

```c
int readFile(char fileName[], float d[], int nmax) {
  int i;
  FILE *fp;
  fp = fopen(fileName,"r");
  if (fp==NULL)
    return 0;
  i = 0;
  while (i<nmax && fscanf(fp, "%f", &d[i])!= EOF)
    i++;
  fclose(fp);
  return i;
}
```

```c
void writeFile(char fileName[], float d[], int n) {
  int i;
  FILE *fp;
  fp = fopen(fileName,"w");
  printf("The numbers are:\n");
  for (i=0; i<n; i++) {
    fprintf(fp,"%f ",d[i]);
  }
  fprintf(fp,"\n");
  fclose(fp);
}
```

Data normalization

Algorithm 2: O(N)

- o Input: array, initialized with a loop
- o Processing: we just need to compute
  - The sum of all values (STOT)
  - For each value, the sum of the i-th value and of its precedent values (sum_i).
  - Based on this, we can compute:
    $p_i = sum_{i-1}/i,$
    $s_i = (STOT-sum_i)/(N-i-1)$
    the normalized values (replacing the original values of the array)
- o Output of the re-computed array.

Data normalization

Example:

Suppose to receive the following six values:

4.0 5.0 3.0 2.0 1.0 7.0

The array of summations sum_i will contain:

4.0 9.0 12.0 14.0 15.0 22.0

STOT = 22.0

Result will be:

- data[0] = (4.0 + 0.0 + (22.0-4.0)/5)/3 = 2.53
- data[1] = (5.0 + 4.0/1 +(22.0-9.0)/4)/3 = 4.08
- data[2] = (3.0 + 9.0/2 +(22.0-12.0)/3)/3 = 3.61
- data[3] = (2.0 + 12.0/3 +(22.0-14.0)/2)/3 = 3.33
- data[4] = (1.0 + 14.0/4 +(22.0-15.0)/1)/3 = 3.83
- data[5] = (7.0 + 15.0/5 + 0.0)/3 = 3.33

Solution - version 2: O(N)

```c
#define NMAX 1000
void normalizeNum(char nfin[],char nfout[]){
  float data[NMAX], sum[NMAX];
  int i, j, N, STOT;
  /* input */
  N = readFile(nfin,data,NMAX); /* input */
  /* partial sums */
  sum[0]=data[0];
  for (i=1; i<N; i++)
    sum[i] = sum[i-1]+data[i];
  STOT = sum[N-1]; /* sum of all numbers */
  ...
```

```c
/* normalize (0 and N-1 handled out of the loop)*/
  data[0] = (data[0] + (STOT-data[0])/(N-1))/3;
  for (i=1; i<N-1; i++) {
    float pred = sum[i-1]/i;
    float succ = (STOT-sum[i])/(N-i-1);
    data[i] = (pred+data[i]+succ)/3;
  }
  data[N-1] = (data[N-1] + sum[N-2]/(N-1))/3;

  writeFile(nfout,data,N); /* output */
}
```

Statistics on groups of data

- Problems are characterized by grouping numerical data into groups that are enumerable and identifiable using an integer value

- The computation can be done on the elements of an array

- Each class or group corresponds to an index value (and to a specific element of the array).

Grouping data

Formulation: write a C function that:

- o receives as parameters an array of integer values in a 0-100 range and the dimension of the array

- o groups the values based on the range of tens they belong to (0-9, 10-19, 20-29, etc.)

- o computes and prints on the screen the occurrence of values belonging to each range.

Grouping data

Example:

Suppose to receive the following 20 values:

3   6   9   16   22   23   30   32   40   48   65   78   7   8   10   15   25   90   27   26

Numbers can be grouped as follows:

- (3, 6, 9, 7, 8), (16, 10, 15), (22, 23, 25, 26, 27), (30, 32), (40, 48), (-), (65), (78), (-), (90), (-)

The output will be:

- 5, 3, 5, 2, 2, 0, 1, 1, 0, 1, 0

Grouping data

- **Solution**:
  - Nested loop (for each range of tens, iterate on all values and count the ones that belong to the given range). Cost is O(n_ranges*n_values)
  - More efficient (O(n_values)): array of counters, taking advantage of the index-value relation. For each input value, we compute the index corresponding to the range of tens the value belongs to, and we update the corresponding counter

- **Data structure**:
  - Array of integers, received as parameter
  - Array of counters (indexes from 0 to 10).

Grouping data

Algorithm:

- Counters all initialized to 0 (ranges of tens are numbered from 0 to 10)

- Loop on input values. Per each value:
  - compute the range it belongs to: integer division
    $$d = data[i]/10;$$
  - using d as index, access the array of counters and increment the corresponding value

- Loop on counters, to print the output

Solution

```c
void countPerTens (int data[], int n){
  int i, d, count[11];
  for (i=0; i<=10; i++)
    count[i]=0;
  for (i=0; i<n; i++) {
    d = data[i]/10;
    count[d] = count[d]+1;
  }
  for (i=0; i<=10; i++)
    printf ("%d data in the tens range %d \n",
            count[i], i+1);
}
```

Encoding problems

ENCODING OF NUMBERS AND TEXTS

Problems of numerical encoding

In numerical encoding problems, arrays can be used to:
- store the digits of a given encoding
- manipulate the values, at the single-digit level

Binary encoding of integer values

Formulation: write a C function that receives as parameter a positive integer value ($0 <= n <= 2^{32} -1$), computes the pure binary encoding and prints the result

Solution:
- Iteratively compute the bits, starting from LSB
- Subsequent divisions by 2, the remainders are the bits of the encoding

Binary encoding of integer values

Data structure:

- one formal parameter (integer): n
- array to store the bits of the encoding

Algorithm:

- Loop to generate the bits, starting from the LSB (at most, 32 bits)
- Loop to print the output, starting from the MSB

Solution

```c
void binaryVector (int n) {
  int i, bit[32];
  i=0;
  do {
    bit[i++] = n%2;
    n = n/2;
  } while (n>0);
  i--;
  while (i>=0) {
    printf("%d",bit[i--]);
  }
  printf("\n");
}
```

Solution

```
void binaryVector (int n) {
  int i, bit[32];
  i=0;
  do {
    bit[i++] = n%2;
    n = n/2;
  } while (n>0);
  i--;
  while (i>=0) {
    printf("%d",bit[i--]);
  }
  printf("\n");
}
```

Computes LSB

Solution

```
void binaryVector (int n) {
  int i, bit[32];
  i=0;
  do {
    bit[i++] = n%2;
    n = n/2;
  } while (n>0);
  i--;
  while (i>=0) {
    printf("%d",bit[i--]);
  }
  printf("\n");
}
```

Repeats until n becomes 0
If initially n=0, computes 1 bit

Problems of text encoding/decoding

Encoding, re-encoding or encryption problems applied to texts, where arrays can be used:

- o to store encoding/decoding tables
- o based on index-value relation
- o as collection of datum – code or code0-code1 pairs
- o as collections of intermediate values.

Encryption of a text file

Formulation:

o Encrypt the content of a text file and write the encrypted text in an output file

o The encoding of the characters are modified based on the encoding table that is stored in a file:

- Each line of the file reports two integer values in a 0 to 255 range, representing the original ASCII code of a character and the ASCII code of the encrypted character, respectively

- The characters whose ASCII code are not reported in the file should not be modified

- The table guarantees that the encryption is univocal

Encryption of a text file

Solution:

- Read the encryption table from the file, build an encryption array where the index corresponds to the original code and the value corresponds to the encrypted code
- Loop to iteratively read characters from the input file
  - Compute the re-encoding of the given character
  - Write the encrypted character in the output file

Data structure:

- Three file pointers `FILE *` to handle input and output file as well as the table file
- String to handle the file names
- Array of characters for the encryption table
- `char` variable to read and encrypt the text

Encryption of a text file

Algorithm:
o Acquisition of file names and file opening
o Initialization of the encryption array (for the characters that should remain unchanged)
o Acquisition of the encryption table from file
o Loop that reads one character at a time from input file, encrypts the character and writes the result in the output file
  • The encryption happens based on the index-value relation of the encryption array

Solution

```c
#define MAXLINE 30
int main(void) {
  char ch, namefile[MAXLINE];
  char table[256];
  FILE *fpin, *fpout, *ftab;
  int i, new;
  printf("name of input file: ");
  scanf("%s", namefile);
  fpin = fopen(namefile,"r");
  printf("name of output file: ");
  scanf("%s", namefile);
  fpout = fopen(namefile,"w");
  printf("name of encryption table file: ");
  scanf("%s", namefile);
  ftab = fopen(namefile,"r");
```

```c
  /* initialize the table: original and
     encrypted code coincide */
  for (i=0; i<256; i++)
    table[i] = (char)i;
  /* read from the encryption table file */
  while (fscanf(ftab,"%d%d",&i,&new)==2)
    table[i] = (char)new;
  /* perform the encryption */
  while (fscanf(fpin, "%c", &ch) == 1) {
    fprintf(fpout,"%c",table[(int)ch]);
  }
  fclose(fpin);fclose(fpout);fclose(ftab);
}
```

Text-processing problems

TEXT PROCESSING: INPUT, MODIFICATION, OUTPUT

Text processing problems

- **Problems that manipulate sequences of characters/strings**

   Examples: input (and interpretation) of a text, modification of a text, generation of messages in a specific format
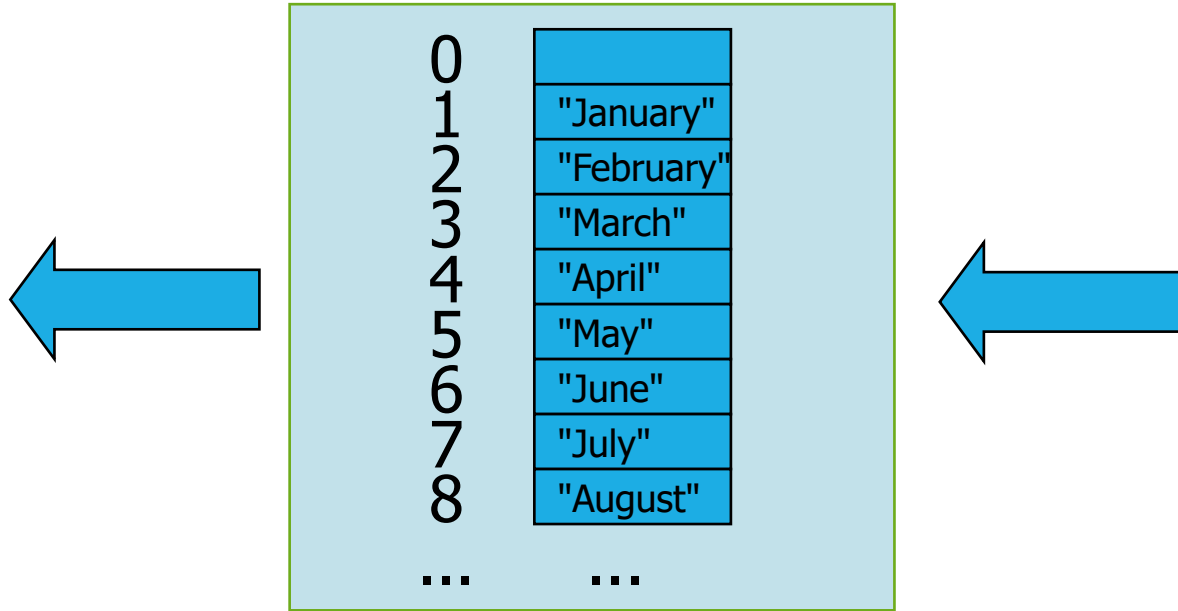
- **Vectors (or matrices) of chars (or strings) may be necessary to:**

   o Generate texts based on specific rules or functions

   o Transform pre-existing texts.

   o Acquire input texts or produce output texts

Vectors of strings and selection

- A vector of strings is stored as a matrix of chars
  - char words[NWORDS][MAXL];

- Problem: given a word (string), search for the word in a vector of words (matrix) to "understand" to which datum (for example, a number) it corresponds

- The selection based on strings requires comparisons (`strcmp`) and `if` conditional constructs (NO `switch`)

- Vectors of strings (used as tables) can allow to translate texts into numerical codes, so that:
  - We can implement the selection with `switch`
  - We can have a better organization of the different cases we need to handle
  - We can have more compact organization of the information, that can be more easily passed to functions or returned by functions (ex. Error codes).
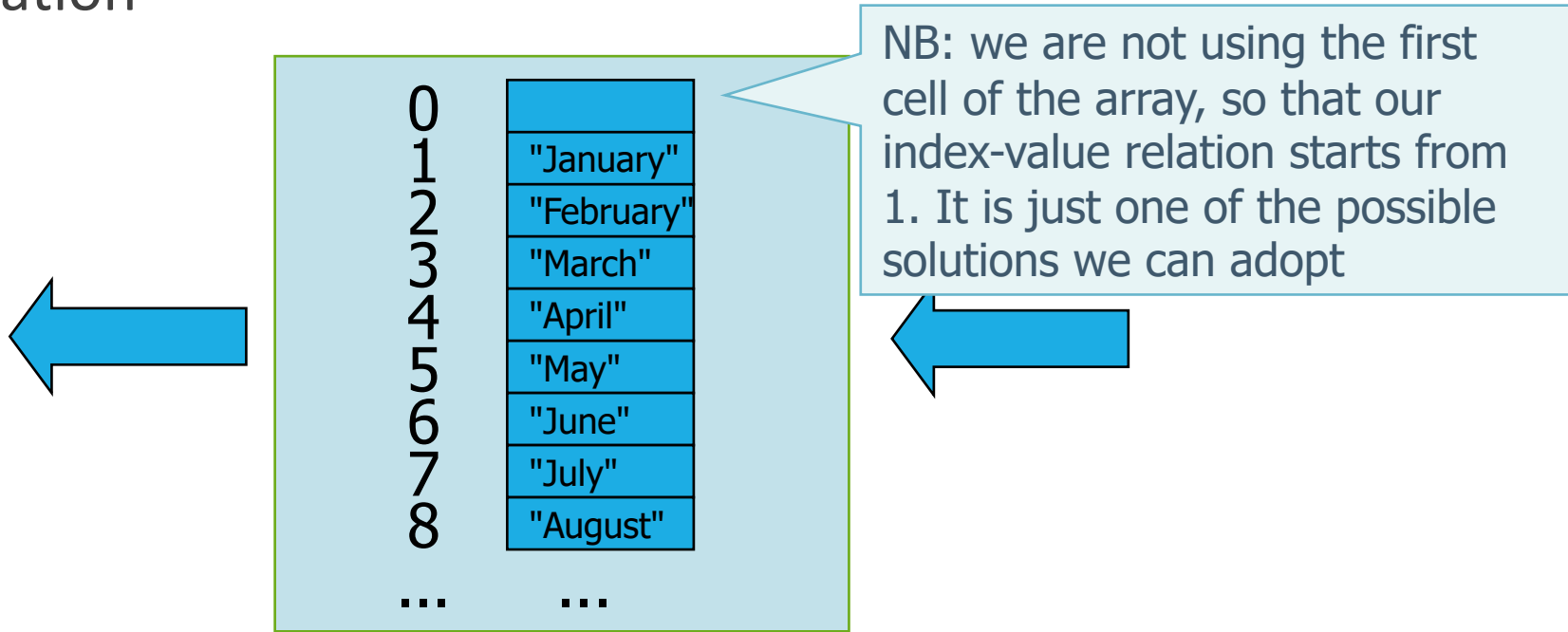
Conversion table (A)

To convert a string to an integer we can use a value-> index relation

0	
1	"January"
2	"February"
3	"March"
4	"April"
5	"May"
6	"June"
7	"July"
8	"August"
…	…

Conversion table (A)

To convert a string to an integer we can use a value-> index relation

0	
1	"January"
2	"February"
3	"March"
4	"April"
5	"May"
6	"June"
7	"July"
8	"August"
...	...

NB: we are not using the first cell of the array, so that our index-value relation starts from 1. It is just one of the possible solutions we can adopt

Conversion table (A)

To convert a string to an integer we can use a value-> index relation

0
1 "January"
2 "February"
3 "March"
4 "April"
5 "May"
6 "June"
7 "July"
8 "August"
... ...

Search for the cell that contains the given name

"March"

Conversion table (A)

To convert a string to an integer we can use a value-> index relation

0
1 "January"
2 "February"
3 "March"
4 "April"
5 "May"
6 "June"
7 "July"
8 "August"
... ...

3

"March"

Return the corresponding index
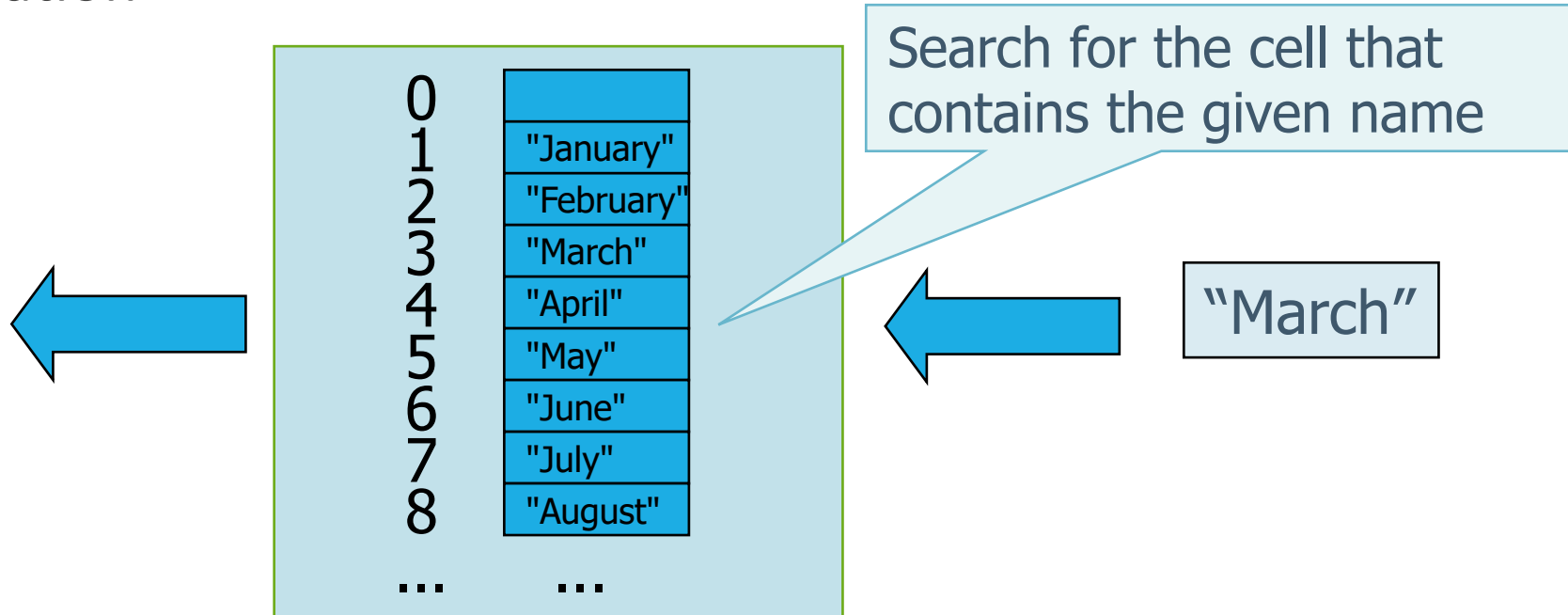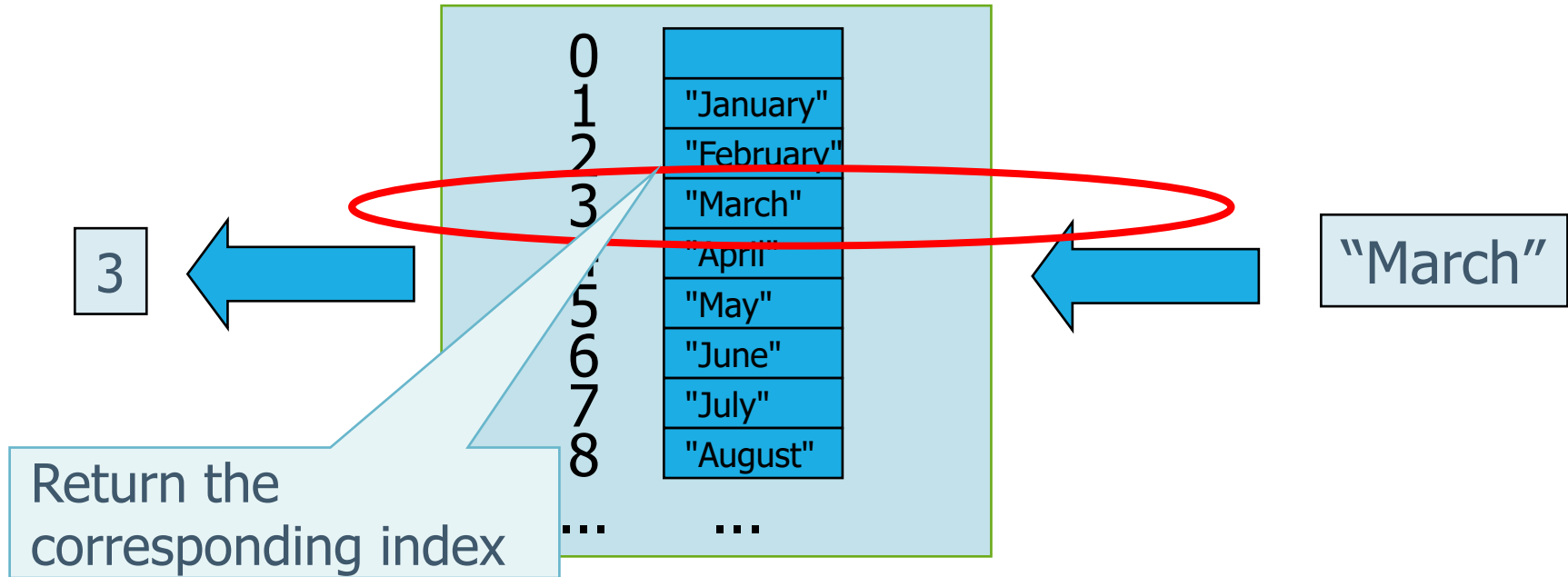
Solution

```
#include <string.h>
int monthStringToNum (char month[]) {
  char table[13][10] = {"",
                        "January","February","March","April","May","June",
                        "July","August","September","October","November","December"};
  int i;
  for (i=1; i<=12; i++) {
    if (strcmp(month,table[i])==0) {
      return i;// found: return index
    }
  }
  return -1; // there is a problem, month not found
}
```
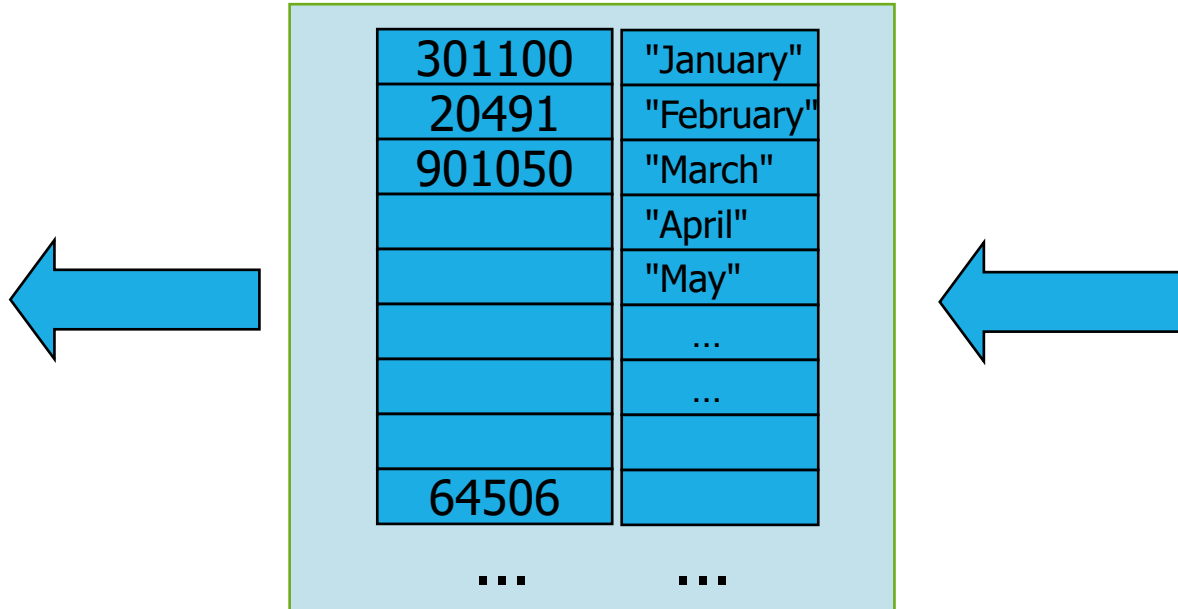
Conversion table (B)

If the integer values are too large (not suitable to be used for indexing the array) we can use a vector of `struct` (code,name), or two parallel vectors
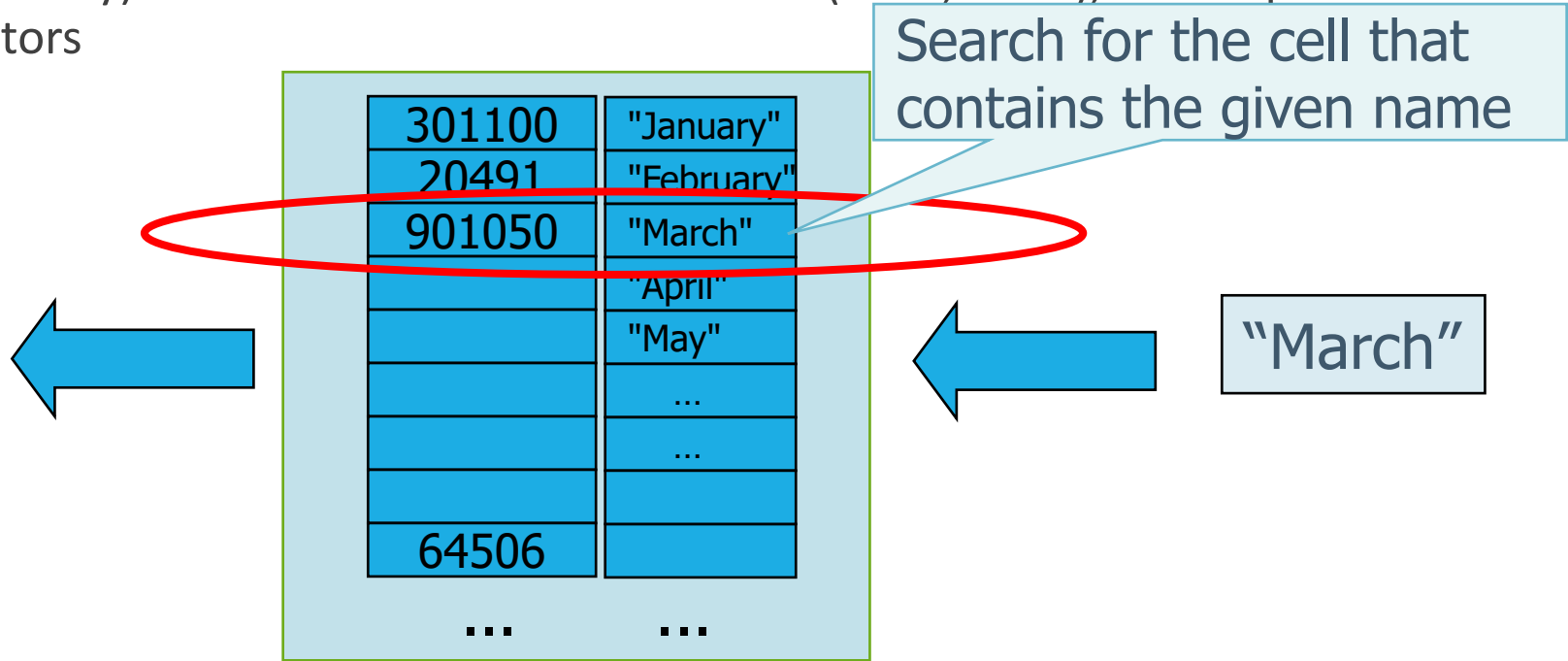
301100	"January"
20491	"February"
901050	"March"
	"April"
	"May"
	...
	...
64506	

...   ...

Conversion table (B)

If the integer values are too large (not suitable to be used for indexing the array) we can use a vector of `struct` (code,name), or two parallel vectors

Search for the cell that contains the given name

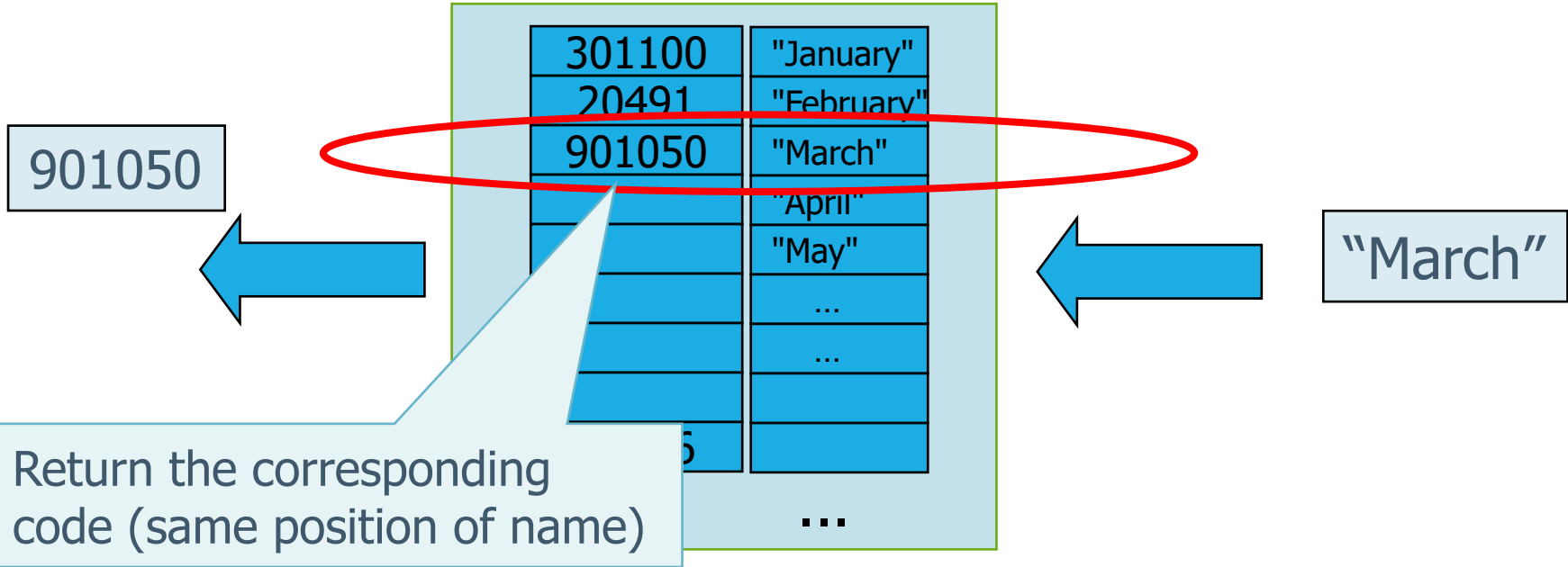|	
301100	"January"
20491	"February"
901050	"March"
	"April"
	"May"
	...
	...
64506	
...	...

"March"

Conversion table (B)

If the integer values are too large (not suitable to be used for indexing the array) we can use a vector of `struct` (code,name), or two parallel vectors

301100	"January"
20491	"February"
901050	"March"
	"April"
	"May"
	...
	...
5	
...	

901050

"March"

Return the corresponding code (same position of name)

Solution

```c
struct monthEntry {
  int num;
  char name[10];
}
int main (void) {
  int i, num;
  char month[10];
  struct monthEntry table[12];
  if (readTable(table) != 0)
    do {
      printf("write a month"); scanf("%s",month);
      n = monthStringToNum(table,month);
      if (n>=0)
        printf("month: %s -> num: %d\n", month, n);
    } while (n>=0);
  return 0;
}
```

```c
int readTable (struct monthEntry t[12]) {
  FILE *fp; int i;
  fp = fopen("table.txt","r");
  if (fp==NULL) {
    printf("Error opening table.txt\n"); return 0;
  }
  for (i=0; i<12; i++)
    fscanf(fp,"%d%s", &t[i].num, t[i].name);
  fclose(fp); return 1;
}
int monthStringToNum (struct monthEntry t[12], char m[]) {
  int i;
  for (i=0; i<12; i++) {
    if (strcmp(m,table[i].name)==0)
      return table[i].num; // found: return num
  }
  return -1; // there is a problem, month not found
}
```

Menu with a string as selector

- Formulation: write a function that iteratively acquires a string from keyboard (50 chars max, may contain spaces):
  - The first sequence of chars different from space is the selector of the menu
    - If the word is "end", the program terminates
    - If the word is one among "seek", "modify", "print" (case-insensitive), the program should call the corresponding seek, modify or print function, passing the rest of the input line as the parameter
  - Any other word should strigger an error message.

- Solution: value-index relation

Menu with a string as selector

- **Modularity**:
  - Input functions and string to code conversion

- **Table:**
  - Array initialized with (pointers at) constant strings

- **String to code conversion:**
  - Loop to search a vector

Solution

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

#define c_seek 0
#define c_modify 1
#define c_print 2
#define c_end 3
#define c_err 4
const int MAXL=51;

int readCommand(void);
void menuWord(void);

void seek(char r[]) { printf("search: %s\n", r); }
void modify(char r[]) { printf("modify: %s\n", r);}
void print(char r[]) { printf("print: %s\n", r); }
```

```c
void strToLower(char s[]) {
  int i,  l = strlen(s);
  for (i=0; i<l; i++)
    s[i]=tolower(s[i]);
}

int main(void) {
  menuWord();
}
```

Solution

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

#define c_seek 0
#define c_modify 1
#define c_print 2
#define c_end 3
#define c_err 4
const int MAXL=51;

int readCommand(void);
void menuWord(void);

void seek(char r[]) { printf("search: %s\n", r); }
void modify(char r[]) { printf("modify: %s\n", r);}
void print(char r[]) { printf("print: %s\n", r); }
```

```
void strToLower(char s[]) {
  int i,  l = strlen(s);
  for (i=0; i<l; i++)
    s[i]=tolower(s[i]);
}

int main(void) {
  menuWord();
```

Definition of constants that make the code more readable (more significant than the corresponding values 0 to 4)

Solution

```
void menuWord(void){
  int command;
  char line[MAXL];
  int i, toContinue=1;
  while (toContinue) {
    command = readCommand();
    fgets(line,MAXL,stdin); /* rest of the line*/
    switch (command) {
      case c_seek: seek(line); break;
      case c_modify: modify(line); break;
      case c_print: print(line); break;
      case c_end: toContinue=0; break;
      case c_err:
      default: printf("wrong command\n");
    }
  }
}
```

```
int readCommand(void) {
  int c;
  char cmd[MAXL];
  char table[4][7] = {
  "seek","modify","print" "end"
  };
  printf("command (seek/modify");
  printf("/print/end): ");
  scanf("%s",cmd); strToLower(cmd);
  c=c_seek;
  while(c<c_err && strcmp(cmd,table[c])!=0)
    c++;
  return (c);
}
```

Solution

```
void menuWord(void){
  int command;
  char line[MAXL];
  int i, toContinue=1;
  while (toContinue) {
    command = readCommand();
    fgets(line,MAXL,stdin); /* rest of the line*/
    switch (command) {
      case c_seek: seek(line); break;
      case c_modify: modify(line); break;
      case c_print: print(line); break;
      case c_end: toContinue=0; break;
      case c_err:
      default: printf("wrong command\n");
    }
  }
}
```

Read command and make name to number conversion

```
  char table[4][7] = {
  "seek","modify","print" "end"
  };
  printf("command (seek/modify");
  printf("/print/end): ");
  scanf("%s",cmd); strToLower(cmd);
  c=c_seek;
  while(c<c_err && strcmp(cmd,table[c])!=0)
    c++;
  return (c);
}
```

Solution

```
void menuWord(void){
  int command;
  char line[MAXL];
  int i, toContinue=1;
  while (toContinue) {
    command = readCommand();
    fgets(line,MAXL,stdin); /* rest of the line*/
    switch (command) {
      case c_seek: seek(line); break;
      case c_modify: modify(line); break;
      case c_print: print(line); break;
      case c_end: toContinue=0; break;
      case c_err:
      default: printf("wrong command\n");
    }
  }
}
```

```
int readCommand(void) {
```

Rest of the line, it is the argument of the command

```
  "seek","modify","print" "end"
  };
  printf("command (seek/modify");
  printf("/print/end): ");
  scanf("%s",cmd); strToLower(cmd);
  c=c_seek;
  while(c<c_err && strcmp(cmd,table[c])!=0)
    c++;
  return (c);
}
```

Solution

```
void menuWord(void){
  int command;
  char line[MAXL];
  int i, toContinue=1;
  while (toContinue) {
    command = readCommand();
    fgets(line,MAXL,stdin); /* rest of the line*/
    switch (command) {
      case c_seek: seek(line); break;
      case c_modify: modify(line); break;
      case c_print: print(line); break;
      case c_end: toContinue=0; break;
      case c_err:
      default: printf("wrong command\n");
    }
  }
}
```

```
int readCommand(void) {

         [4][7] = {
    ,"modify","print" "end"
  };
  printf("command (seek/modify");
  printf("/print/end): ");
  scanf("%s",cmd); strToLower(cmd);
  c=c_seek;
  while(c<c_err && strcmp(cmd,table[c])!=0)
    c++;
  return (c);
}
```

Select the function to execute

Solution

```
void menuWord(void){
  int command;
  char line[MAXL];
  int i, toContinue=1;
  while (toContinue) {
    command = readCommand();
    fgets(line,MAXL,stdin); /* rest of the line*/
    switch (command) {
      case c_seek: seek(line); break;
      case c_modify: modify(line); break;
      case c_print: print(line); break;
      case c_end: toContinue=0; break;
      case c_err:
      default: printf("wrong command\n");
```

Name to number conversion

```
}
```

```
int readCommand(void) {
  int c;
  char cmd[MAXL];
  char table[4][7] = {
  "seek","modify","print" "end"
  };
  printf("command (seek/modify");
  printf("/print/end): ");
  scanf("%s",cmd); strToLower(cmd);
  c=c_seek;
  while(c<c_err && strcmp(cmd,table[c])!=0)
    c++;
  return (c);
}
```

MENU: variant with enum type

- **enum type in C**:
  - Automatically assiciate names to integer values starting from 0
  - Ex.: `enum trafficlight {green,red,yellow};`
    Defines a new type "enum trafficlight", that associates the names green, red and yellow to values 0, 1, 2 respectively
  - It is possible to define different relations (omitting some values), <u>we will never do it</u>
  - Careful: in C you can apply arithmetic operators, in C++ NO
  - Often used with typedef
    Ex.: `typedef enum {green,red,yellow} trafficLight;`

- `In the menu, we can use enum instead of defining the numerical constants one by one with a #define`

Solution

```
typedef enum {
  c_seek,c_modify,c_print,c_end,c_err
} t_commands;
...
void menuWord (void){
  t_commands command;
  char line[MAXL];
  int i, toContinue=1;
  while (toContinue) {
    command = readCommand();
    fgets(line,MAXL,stdin); /* rest of the line*/
    switch (command) {
      case c_seek: seek(line); break;
      ...
    }
  }
}
```

```
t_commands readCommand(void) {
  t_commands c;
  char cmd[MAXL];
  char table[4][7] = {
  "seek","modify","print" "end"
  };
  printf("command (seek/modify");
  printf("/print/end): ");
  scanf("%s",cmd); strToLower(cmd);
  c=c_seek;
  while(c<c_err && strcmp(cmd,table[c])!=0)
    c++;
  return (c);
}
```

Processing texts character by character

- A text can be generated or modified at the character level. A vector or matrix can be used:
  - to represent the input text
  - to temporarily store or manipulate a string or a matrix of characters

Processing texts character by character

- Manipulation of word or sentences:
  - A word (or a sentence) can be processed one character at a time
  - In some cases, we need an array to directly access the characters

Examples:

- Palindromy verification
- Cut and paste a portion of a string to a different location
- search/replace a substring

Verfication of palindromy

- Formulation: write a C function that checks whether a string, received as parameter, is palindrome (difference between upper and lowercase characters should not be taken into account) :
  - A word is palindrome if it is the same when it is read left to right or right to left

    Examples: Anna, madam, otto, abcdefgFEDCBA

- Algorithm:
  - Iteratively compare corresponding characters (0 – N-1, 1 – N-2, etc …)

- Data structure:
  - The array (string), received as parameter
  - Two integers, for the indexes of the charaters to be compared
  - A flag variable to terminate the iterations

Solution

```
int palindrome (char word[]) {
  int i, n, pal=1;

  n = strlen(word);
  for (i=0; i<n/2 && pal; i++) {
    if (toupper(word[i]) != toupper(word[n-1-i]))
      pal = 0;
  }
  return pal;
}
```

Generation of figures/graphs

Figures or graphs can be interpreted as a matrix of printable characters (25 rows and 80 columns):

- o This allows to print the figure without the necessity of following the sequentiality of the output (either on video or on a text file)
- o The figure to be printed can be represented as a matrix of characters and then printed on the screen/file

Visualization of a parabola

- **<u>Formulation:</u>**
  - Given a parabole of equation
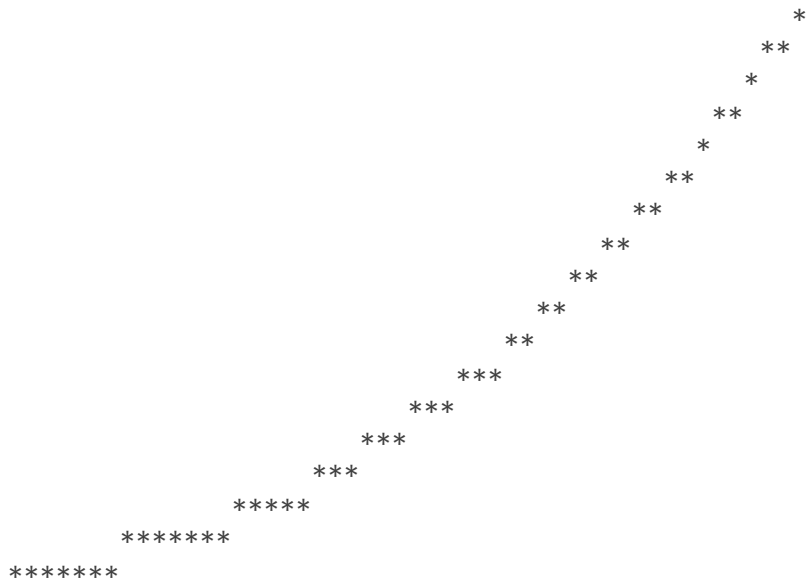
  $$y = ax^2 + bx + c = 0$$

  - Write a C program to:
    - Acquire from keyboard the parabola coefficients a, b, c, and the values of the ranges (xmin, xmax) and (ymin, ymax), respectively for the x-axis and the y-axis
    - print, in a 20x80 area, a graph (with horizontal axis corresponding to x-axis) representing the parabola within the intervals [xmin,xmax], [ymin,ymax]

Visualization of a parabola

Example: if we acquire the following values:

a=1.0, b=2.0, c=1.0, x0=-1.0, xn=4.0, ymin=-1.0, ymax=10.0 the output should be:

```
                                          *
                                        **
                                       *
                                     **
                                    *
                                  **
                                **
                              **
                            **
                          **
                        **
                      **
                    ***
                  ***
                ***
              ***
            *****
          *******
        *******
```

Visualization of a parabola

- <u>Data structure</u>: we need scalar data to represent
  - coefficients: a, b, c (`float`)
  - intervals: xmin, xmax, ymin, ymax (`float`)
  - Intermediate values: stepX stepY (length of intervals), x, y (`float`)
  - Indexes : i, j (`int`)

We need a matrix (`of type char`) to represent the graph.

Visualization of a parabola

- **<u>Algorithm</u>:**
  - input data and compute steps (= length of intervals)
  - Initialize the matrix with space characters
  - Iterate on x values
    - compute y(x)
    - If it is in the range [ymin,ymax] convert to integer and assign '*' in the matrix
  - Iterate on rows and columns to print the output matrix

Solution

```c
#include <stdio.h>
#include <math.h>
const int NR=20, NC=80;
int main(void) {
  float a, b, c, x, y, stepX, stepY,
        xmin, xmax, ymin, ymax;
  int i, j;
  char page[NR][NC];
  FILE *fpout = fopen("out.txt","w");
  printf("Coefficients (a b c): ");
  scanf("%f%f%f",&a,&b,&c);
  printf("X axis range (xmin xmax): ");
  scanf("%f%f",&xmin,&xmax);
  printf("Y axis range (ymin ymax): ");
  scanf("%f%f",&ymin,&ymax);
```

```c
/* matrix initialization*/
for (i=0; i<NR; i++)
  for (j=0; j<NC; j++)
    page[i][j] = ' ';
stepX = (xmax-xmin)/(NC-1);
stepY = (ymax-ymin)/(NR-1);
/* compute parabola points */
for (j=0; j<NC; j++) {
  x = xmin + j*stepX;
  y = a*x*x + b*x + c;
  if (y>=ymin && y<=ymax) {
    i = (y-ymin)/stepY;
    step[i][j] = '*';
  }
}
```

Solution

```
/* print matrix row by row:
   the minimum value of y (first row) needs to be printed last */
for (i=NR-1; i>=0; i--) {    // per each value of y
  for (j=0; j<NC; j++)         // per each value of x
    fprintf(fpout,"%c",page[i][j]);
  fprintf(fpout,"\n");
}
fclose(fpout);
}
```

Processing texts one string at a time

Text can be processed at the string level when:

- o it is possible to identify substrings (sequences of characters) on which we can apply atomic operations
- o the atomic operations should use functions (either library functions or functions implemented by the programmer).

We may need arrays to generate or temporarily store the strings to be processed.

Text formatting

- **Formulation**:
  - A text file is given, where the lines can be decomposed into words of no more than 20 char each, with space, '\t' or '\n' characters as separators
  - Write a C function that reads the file (first parameter of the function) and copies the content in another file (second parameter), after modifying it as follows:
    - Sequences of more than one space should be reduced to one single space
    - Each line should have a maximum lenght of lmax (third parameter of the function). To do so, the function should either add '\n' in substitution of spaces or eliminate '\n', so that the lenght is of each line is the maximum possible length that is smaller than the given threshold.
    - The text should be centered with respect to lmax.

Text formatting

- Solution:
  o The solution is similar to the one of the lectures C2 (Problem solving with scalar data), except for the text centering
  o To center a text we need to store the entire line before printing it (so that we can compute the number of initial spaces to be printed) :
    - We can use an array as a buffer (temporary storage)
    - To center in a `lmax`-wide area a line of `l` characters, we need to print `(lmax-l)/2` spaces before printing the line

Solution

```
#include <string.h>
const int STRLEN=21;
const int LMAX=255;
...
void format(char nin[],char nout[],int lmax){
  FILE *fin=fopen(nin,"r");
  FILE *fout=fopen(nout,"w");
  char word[STRLEN], line[LMAX]= "";
  int i,l;
  l=0;
  while (fscanf(fin,"%s", word)==1) {
    if (l+1+strlen(word) <= lmax) {
      strcat(line," "); strcat(line,word);
      l+=1+strlen(word);
    }
```

```
    else {
      for (i=0; i<(lmax-l)/2; i++)
        fprintf(fout," ");
      fprintf(fout,"%s\n",line);
      strcpy(line,word);
      l=strlen(word);
    }
  }
}
```

Verification and selection problems

VERIFICATION, SELECTION AND SORTING APPLIED TO ARRAYS

Verification and selection problems

- Verification consists in deciding whether a given set of information or data are compliant with certain acceptance criteria

- Selection means to verify the data and select (choose) the ones that are correspondent to the verification criteria

- Search is a possible selection modality:
  - We often search the value that corresponds to acceptance criteria
  - May involve large number of data

- Arrays may be used:
  - To store the set of values on which we need to apply the verification criteria
  - To store the selected data

Verification on sequences

- To verify a sequence means to decide whether the sequence is compliant with given acceptance criteria

- An array may be necessary if the acceptance criteria requires to process all the data of the sequence.

Verification of repetitions

- **Formulation:**
  - A text file contains a sequence of numerical values (real)
    - The first line reports a positive integer value that specifies the number of values of the sequence
    - The following lines report the values, with spaces or return as separator
  - Write a C function that receives a pointer to the input file (already opened) as parameter and verifies if all the values are repeated at least once in the sequence
    - A value is considered repeated if there is at least another value of the sequence so that their absolute difference is lesser than 1%

Verification of repetitions

- Solution:
  - Analyze the data with a nested loop. Per each value, verify that there is at least another value that is considered a repetition (that is, the absolute difference of the two values is ≤ 1% of the maximum absolute value of the two

- Data structure:
  - Array to contain the values read from the file
  - Scalar variables: indexes, counter and flag

- Algorithm:
  - Store data in an over-sized array (static allocation)
  - Verification with nested loop
  - Computation with use of flag.

Solution

```
int repeatedData(FILE *fp) {
  float data[MAXDATA];
  int ndata, i, j, repeated;
  fscanf(fp,"%d",&ndata);
  for (i=0; i<ndata; i++)
    fscanf(fp,"%f",&data[i]);
  for (i=0; i<ndata; i++) {
    repeated = 0;
    for (j=0; j<ndata && !repeated; j++)
      if (i!=j && similar(data[i],data[j]))
        repeated=1;
    if (!repeated) return 0;
  }
  return 1;
}
```

```
int similar (float a, float b) {
    if (fabs(a)>fabs(b))
      return (fabs(a-b)/fabs(a) < 0.01);
    else
      return (fabs(a-b)/fabs(b) < 0.01);
}
```

Selection of data

- While verifying a set of data, we may want to segregate (select) the values that correspond to the verification criteria

- A variant of a verification problem:
  o Data first undergo verification
  o Then, the values that pass the acceptance criteria are selected

- A search is a specific variant of a selection problem:
  o We select the value (if it exists) according to certain search criteria

Matricola→name conversion

- <u>Formulation</u>:
  - Write a function that is able to determine the name of a student starting from the matricola ID (given as first parameter of the function)
  - As the matricola numbers are large (6 digits, MMAX) it is not opportune to build a solution on top of the index-value relation of an array. We represent matricola IDs as strings. Suppose the name as a maximum length NMAX.

- The conversion table, second parameter of the function, is an array of `struct`, where each struct has the matricola ID and the name of a student as fields. The length of the array (i.e. of the conversion table) is given as third parameter to the function.

- The fourth parameter is a string where to store the obtained result

Matricola→name conversion

- **Solution**:
  - Iterate on the data of the array that contains the conversion table, comparing each matricola ID with the requested one

- **Data structure**:
  - The table is an array (given as parameter)
  - Matricola ID and corresponding output name (parameters) are strings

- **Algorithm**:
  - The search consists in a data verification problem
  - The function searches for the required matricola id and stores the corresponding name in the appropriate string.
  - The function returns 1 or 0 to indicate whether the search was successful

Solution

```c
#include <string.h>
typedef struct {
  char matricola[MMAX+1], name[NMAX+1];
} t_stud;
...
int matrName(char m[], t_stud table[], int n, char n[]){
  int i;
  for (i=0; i<n; i++) {
    if (strcmp(m,table[i].matricola)==0) {
      strcpy (n, table[i].name);
      return 1;
    }
  }
  return 0;
}
```

Solution

```
#include <string.h>
typedef struct {
  char matricola[MMAX+1], name[NMAX+1];
} t_stud;
...
int matrName(char m[], t_stud table[], int n, char n[]){
  int i;
  for (i=0; i<n; i++) {
    if (strcmp(m,table[i].matricola)==0) {
      strcpy (n, table[i].name);
      return 1;   // the conversion was successful
    }
  return 0;  // the conversion was unsuccessful
}
```

Typedef is used to define a synonym of (**t_stud**) of the type **struct**.

Sorting problems

- Sorting consists in generating a permutation of a sequence of data so that the resulting sequence is compliant with some ordering criteria

- Total sorting is often applied on arrays, that is suitable to store linear successions of ordered data, where the elements' values increase (or decrease) at incresing value of the index
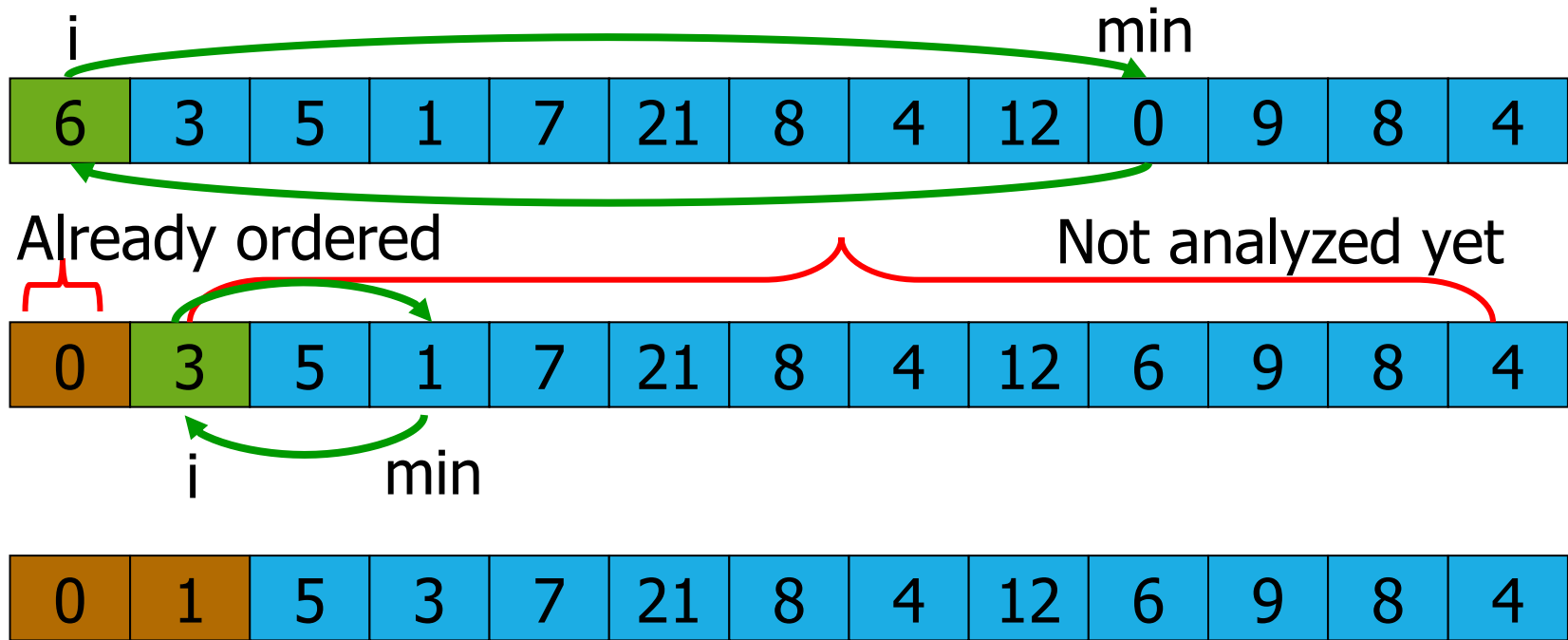
Selection sort

- Formulation: write a C function that:
  - receives an array of integer values and its dimension as parameters
  - sorts the values in increasing order, applying a **selection sort** algorithm

- Solution: selection sort
  - Sorting based on repeated search/selection of minimum value

Selection sort

- Data structure and algorithm:
  - data: array A of N integers (A[0] ... A[N-1]). It can be divided into 2 subvectors:
    - left: ordered
    - right: unordered
  - By definition, if N is 1 the vector is ordered
  - Incremental approach:
    - step i: the miminum value of subvector (A[i] ... A[N-1]) is assigned to A[i]; increment i
  - Termination: all elements are ordered.

Example

Solution

```
void selectionSort (int A[], int N) {
  int i, j, imin, temp;


  for (i=0; i<N-1; i++) {
    /*find the index of the min in A[i]..A[N-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (A[j] < A[imin]) imin = j;
    /*swap min with A[i]*/
    temp = A[i];
    A[i] = A[imin];
    A[imin] = temp;
  }
}
```

Solution

```
void selectionSort (int A[], int N) {
  int i, j, imin, temp;

  for (i=0; i<N-1; i++) {
    /*find the index of the min in A[i]..A[N-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (A[j] < A[imin]) imin = j;
    /*swap min with A[i]*/
    temp = A[i];
    A[i] = A[imin];
    A[imin] = temp;
  }
}
```

External loop, N-1 iterations

Solution

```
void selectionSort (int A[], int N) {
  int i, j, imin, temp;

  for (i=0; i<N-1; i++) {
    /*find the index of the min in A[i]..A[N-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (A[j] < A[imin]) imin = j;
    /*swap min with A[i]*/
    temp = A[i];
    A[i] = A[imin];
    A[imin] = temp;
  }
}
```

Internal loop, N-i-1 iterations

Solution

```
void selectionSort (int A[], int N) {
  int i, j, imin, temp;


  for (i=0; i<N-1; i++) {
    /*find the index of the min in A[i]..A[N-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (A[j] < A[imin]) imin = j;
    /*swap min with A[i]*/
    temp = A[i];
    A[i] = A[imin];
    A[imin] = temp;
  }
}
```

"in loco" algorithm: swaps locally the elements, without requiring a second array as support

Sorting an array of struct

- Example: function `sortStudents` defined, but NOT implemented yet

- Une of the fields used as the ordering key (comparison)
  - `struct student`, field score

- It is very convenient to implement a function that handles the comparison
  - `STUDlt` (less than), or `STUDge` (greater or equal), …

Solution

```
void sortStudent(struct student el[], int n) {
  int i, j, imin;
  stuct student temp;

  for (i=0; i<n-1; i++) {
    /*find index of min in el[i]..el[n-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (el[j].score < el[imin].score) imin = j;
    /*swap min with el[i]*/
    temp = el[i];
    el[i] = el[imin];
    el[imin] = temp;
  }
}
```

Solution (with external function for the comparison)

```
void sortStudent(struct student el[], int n) {
  int i, j, imin;
  stuct student temp;

  for (i=0; i<n-1; i++) {
    /*find index of min in el[i]..el[n-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (STUDlt(el[j],el[imin])) imin = j;
    /*swap min with el[i]*/
    temp = el[i];
    el[i] = el[imin];
    el[imin] = temp;
  }
}
```

```
/* comparison: retuns !=0 (true) if the score of s1 is
lesser than the one of s2, 0 (false) otherwise */

int STUDlt (struct student s1, struct student s2) {
  return (s1.score < s2.score);
}
```

Sorting of vectors of strings

- A vector of strings is a matrix of char
- To handle the matrix as a vector:
  - Use the first index to identify the row
  - Use strcmp to compare rows (strings)
  - Use strcpy to assign/copy strings

Solution

```c
void sortNames(char names[][MAXL], int n) {
  int i, j, imin;
  char temp[MAXL];

  for (i=0; i<n-1; i++) {
    /*find index of min in names[i]..names[n-1]*/
    imin = i;
    for (j = i+1; j < N; j++)
      if (strcmp(names[j],names[imin])<0) imin = j;
    /*swap min with names[i]*/
    strcpy(temp,names[i]);
    strcpy(names[i],names[imin]);
    strcpy(names[imin],temp);
  }
}
```

Solution

```c
void sortNames(char names[][MAXL], int n) {
    int i, j, imin;
    char temp[MAXL];

    for (i=0; i<n-1; i++) {
        /*find index of
        imin = i;
        for (j = i+1; j
            if (strcmp(na
        /*swap min with
        strcpy(temp,nam
        strcpy(names[i]
        strcpy(names[imin],temp);
    }
}
```

The function does not need to know the number of rows of the matrix. Only the caller NEEDS to know this dimension. Advantage: the function can be used with matrix of different dimensions.
NB: THIS IS VALID ONLY FOR THE NUMBER OF ROWS!
THE NUMBER OF COLUMNS NEEDS TO BE FIXED

Solution

```
void sortNames(char names[][MAXL], int n) {
  int i, j, imin;
  char temp[MAXL];

  for (i=0; i<n-1; i++) {
    /*find index of
    imin = i;
    for (j = i+1; j
      if (strcmp(nam
    /*swap min with
    strcpy(temp,name
    strcpy(names[i],names[imin]);
    strcpy(names[imin],temp);
  }
}
```

The function needs to know how many rows of the matrix are used and need to be processed by the sorting algorithm.
The caller function passes this additional information as the second parameter.