Unit 5

Arrays, Pointers, and Strings

Alberto Casagrande
Email: acasagrande@units.it

a.a. 2019/2020

The Italian Lottery and Late Numbers

The Italian Lottery (Superenalotto) is gamble game in which you have to guess 6 numbers randomly selected without repetition among all the naturals from 1 to 90.

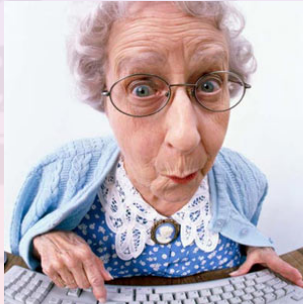
The game has not memory: the probability for a number to be one among the winning 6 is the same at each repetition of the game

The Italian Lottery and Late Numbers

Despite this, . . .

The Italian Lottery and Late Numbers

Despite this, ... someone keeps betting on late numbers and ask you a program to track them.



A Program for Late Numbers

```
int delay_of_1=0,...,delay_of_90=0;
...

if (selected_number!=1)
  delay_of_1++;
else
  delay_di_1=0;


...

if (selected_number!=90)
  delay_of_90++;
else
  delay_of_90=0;
...
```

Issues in Previous Code

- not really flexible
- extremely redundant

C functions do not help in this case because we want to update a data structure, not evaluate a function.

Issues in Previous Code

- not really flexible
- extremely redundant

C functions do not help in this case because we want to update a data structure, not evaluate a function.

It would be nice to select the variable `delay_of_<number>` according to `<number>`

Something like `delay_of_[selected_number]`

Arrays

An array is an indexed data structure to store values having all the same type.

E.g.,

A=

3	1	-1	5	0
0	1	2	3	4

`A[i]` refers to the `i`th element of array `A`

Array Usage

Just like any other variable, array must be declared before usage.

The syntax is

```
...
<elements type> <array name>[<length>];
...
```

Array Usage

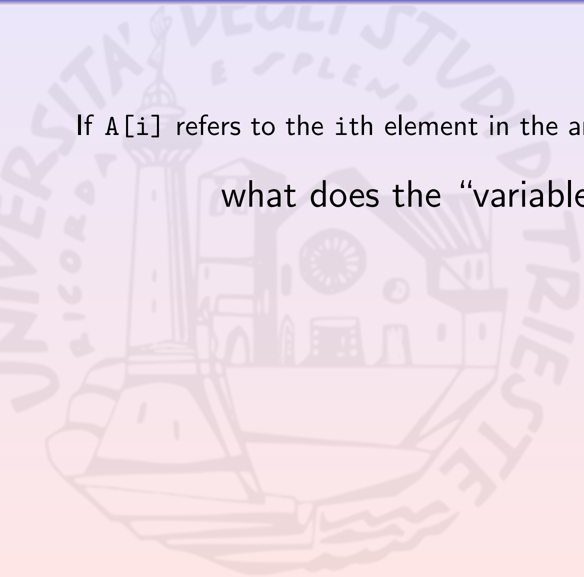
```
...
int delay[90]; /* the first index is 0 */

for (int i=0; i<90; i++) /* init delays */
  delay[i]=0;

... /* later */

for (int i=0; i<90; i++)
  delay[i]++;

for (int j=0; j<6; j++)
  delay[selected[j]-1]=0; /* selected numbers
                              have delay 0 */
...
```

Array Initialization

The C programming language does not initialize array elements.

If initialization is required, you have to take care of it.

Arrays and Their Elements



If `A[i]` refers to the `i`th element in the array A,

what does the "variable" `A` store?

Arrays and Their Elements

If `A[i]` refers to the `i`th element in the array A,

what does the "variable" `A` store?

`A` maintains the memory address of `A` itself.

Try the instruction `printf("%ld", A);`

Arrays and Functions

Arrays can also be used as function parameters

```
void test(int A[]) { A[0]=0; }

int main(int argc, char *argv[])
{
  int A[10]; A[0]=1; test(A);

  printf("%d\n", A[0]);

  return 0;
}
```

Pointers

Are variables whose values are memory addresses of some data structure.

They must be declared by using the syntax:

```
<pointed type>* <pointer name>;
```

`<pointed type>*` a new type for every `<pointed type>`.

Pointers

A pointer of any variable `v` can be obtained by `&v`.

The content of what pointed by `p` can be accessed by `*p`.

E.g.,

```
int v=0; int* p=&v;

*p = *p + 1;

printf("%d\n", v);
```

Pointers as Parameters

Pointers can be used as parameters too.

```
void d_value(int* v) { *v=2*(*v); }

int main(int argc, char *argv[])
{
  int a; a=1; d_value(&a);

  printf("%d\n", a);

  return 0;
}
```

Reading Data from the Standard Input

Pointer parameters allow to “scan” data from the stdin.

```
int main(int argc, char *argv[])
{
  int a; char c; float f;

  scanf("%d %c %f", &a, &c, &f);
  printf("%d %c %f\n", a, c, f);

  return 0;
}
```

C Pointer Arithmetic

C allows to handle pointers like natural numbers.

We can sum and subtract natural values to pointers.

E.g.,

```
int *a;
...
a = a + 2;
...
```

C Pointer Arithmetic

C allows to handle pointers like natural numbers.

We can sum and subtract natural values to pointers.

E.g.,

```
int *a;
...
a = a + 2;
...
```

The results may seem to be weird at first sight.

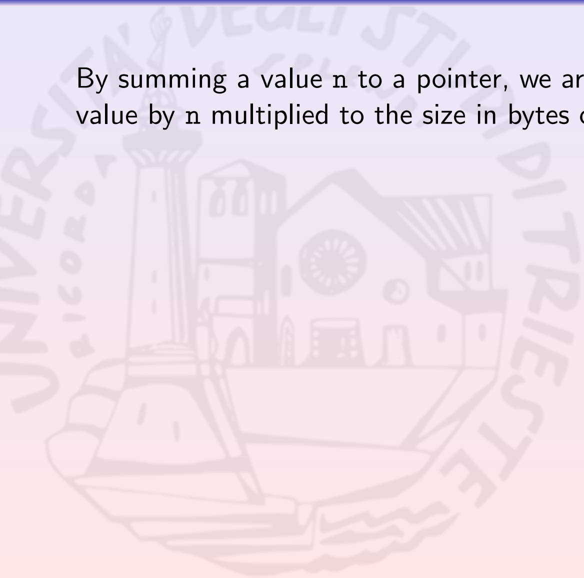
Examples of Pointer Arithmetic

```
int main(int argc, char *argv[]) {
  int *a=0;

  printf("%ld %ld %ld\n",
         (long int)( ((int    *)a) + 1),
         (long int)( ((char   *)a) + 1),
         (long int)( ((double *)a) + 1));

  return 0;
}
```

The execution returns

```
4 1 8
```

Semantics of Pointer Arithmetic in C



By summing a value `n` to a pointer, we are increasing the pointer value by `n` multiplied to the size in bytes of the pointed type.

Semantics of Pointer Arithmetic in C

By summing a value `n` to a pointer, we are increasing the pointer value by `n` multiplied to the size in bytes of the pointed type.

Semantics of Pointer Arithmetic in C

E.g., If int is a 4 bytes type and `p` is an `int` pointer, then `p+1` is equal to the address in `p` plus 4.

More in general, if `p` has type `t*`, then

```
p + n == ((unsigned long int)p) + n * sizeof(t)
```

Pointers vs Arrays

Due of their arithmetic, pointers can be used in place of arrays.

Array elements can also be accessed by a “pointer-like” syntax.

```
int A[] = {0,1,2}; /* initialise the array */
int* p = A;

printf("%d %d\n", A[0], p[0]);
printf("%d %d\n", *(A+2), *(p+2));
```

Pointers vs Arrays

However, pointers and arrays are not equivalent!

Pointers can be re-assigned, arrays cannot.

```
int A[] = {0,1,2}; int B[];
int *p;

p=A+2; /* this is valid */

A=A+2; /* this is not valid */
B=A; /* this is not valid too */
```

Can You Now Guess . . .

. . . the type of b in

```
void test(char *b[])...
```

Can You Now Guess . . .

. . . the type of b in

```
void test(char *b[])...
```

Have you evere seen something similar?

Can You Now Guess . . . (Cont'd)

```
int main(int argc, char *argv[]) {
   return 0;
}
```

argv is an "array" of arrays of chars.

What can an array of characters represent?

Can You Now Guess . . . (Cont'd)

```
int main(int argc, char *argv[]) {
   return 0;
}
```

argv is an "array" of arrays of chars.

What can an array of characters represent? Strings!!!

argv stores the arguments of the command line execution.

Parameters of the main Function

```
int main(int argc, char *argv[]) {
  for (int i=1; i< argc; i++)
    printf("Param #%d is %s\n", i, argv[i]);

  return 0;
}
```

Execute the compiled program by using the command

```
./a.out test -3 5.4
```

Do You Remember Escape Sequences?

- `\n` newline
- `\b` backspace
- `\t` horizontal tabulation
- `\\` backslash character
- `\"` double quotation character
- `\a` alert

The string terminator should always end the string in the array of characters!

Do You Remember Escape Sequences?

- `\n` newline
- `\b` backspace
- `\t` horizontal tabulation
- `\\` backslash character
- `\"` double quotation character
- `\a` alert
- `\0` string terminator

The string terminator should always end the string in the array of characters!

Strings in C

The program

```
int main(int argc, char *argv[]) {
    char a_str[]="\"It\" \n \beni\bds \0 here";

    printf("%s %c\n", a_str, a_str[3]);

    return 0;
}
```

outputs

```
"It"
ends "
```

Some Useful Functions on Strings

Include `string.h` and use the following functions

`strcat` joins two strings

```
char* strcat (char *s1, const char *s2);
```

s1 and the result will "contain" s1.s2

`strcmp` compares two strings

```
int strcmp (const char *s1, const char *s2);
```

the result will be < 0 if `s1` is smaller than `s2`, > 0 if it is greater, and $= 0$ if they are the same.

Some Useful Functions on Strings (Cont'd)

`strcpy` copies a string

```
char* strcpy (char *dst, const char *src);
```

`strlen` gets the length

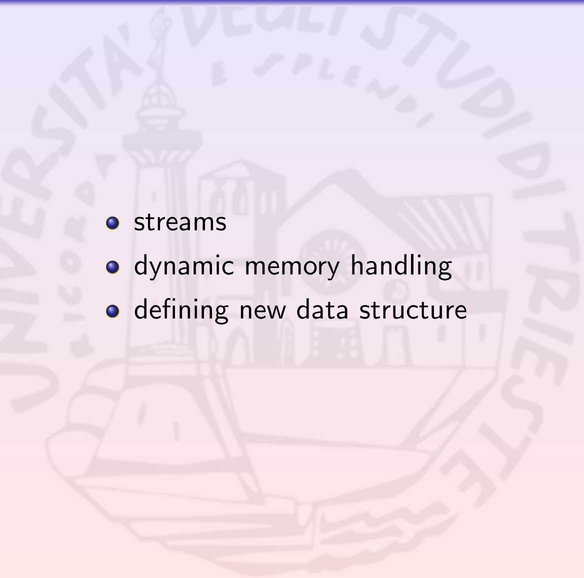
```
size_t strlen (const char *s);
```

`size_t` is a unsigned numerical type to represent data structure size

Strings in C

```c
char a_str[100]="Let me see ";
char b_str[100]="how it works";
char c_str[100];

strcpy(c_str, a_str);
printf("cpy %s\n", c_str);

strcat(c_str, b_str);
printf("cat %s\n", c_str);

printf("cmp %d %d %d", strcmp(a_str, b_str),
                       strcmp(a_str, a_str),
                       strcmp(b_str, a_str));
printf("len %lu\n", strlen(a_str));
```

Coming soon. . .



- streams
- dynamic memory handling
- defining new data structure