Introduction to Computer and Programming

7. Arrays and pointers

Manuel – Summer 2019

Chapter organisation

Basic array manipulations

In C an array is defined by three parameters:

- A name
- The data type of its elements
- A size, i.e. the number of elements compositing it

Example.

```
1  int a[4]={1,2,3,4};
```

Basic array manipulations

In C an array is defined by three parameters:

- A name
- The data type of its elements
- A size, i.e. the number of elements compositing it

Example.

```c
int a[4]={1,2,3,4};
```

Simple manipulations:

- Set the first element to 0
- Add 1 to the second element
- Set the third element to the sum of the third and fourth
- Display all the elements

```c
a[0]=0;
a[1]++;
a[2]+=a[3];
for (i=0; i<4;i++)
  printf("%d\n",a[i]);
```

Arrays and functions

array_fct.c

```c
#include <stdio.h>
double average(int arr[], size_t size);
int main () {
  int elem[5]={1000, 2, 3, 17, 50};
  printf("%lf\n",average(elem,5));
}
double average(int arr[], size_t size) {
  unsigned long i;
  double avg, sum=0;
  for (i = 0; i < size; ++i) {
    sum += arr[i];
  }
  avg = sum / size;
  return avg;
}
```

Arrays and functions

Understanding the code:

- Why is the prototype of the function `average` mentioned before the `main` function?
- How to pass an array to a function?
- Is the size of an array automatically passed to a function?
- When passing an array to a function how to ensure the function knows its size?

From one die to two dice

Understand the following code and adapt it to handle two dice

die.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIDES 6
#define ROLLS 1000
int main () {
  int i, tab[SIDES];
  srand(time(NULL));
  for (i=0; i < SIDES;  i++) tab[i]=0;
  for (i=0; i < ROLLS;  i++) tab[rand()%SIDES]++;
  for (i=0;i<SIDES;i++) printf("%d (%d)\t",i+1,tab[i]);
  printf("\n");
}
```

In the previous code, how is the array initialized?

More dice

dice.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define DICE 4
#define SIDES 10
#define ROLLS 100000
int main () {
  int i, j, t, res[DICE*SIDES-DICE+1]={0};
  srand(time(NULL));
  for (i=0; i < ROLLS; i++) {
    t=0;
    for(j=0;j<DICE;j++) t+=rand()%SIDES;
    res[t]++;
  }
  for (i=0;i<DICE*SIDES-DICE+1;i++) {
    printf("%d (%d)  ",i+DICE,res[i]);
  }
  printf("\n");
}
```

More dice

Understanding the code:

- How is the array initialized?
- What is `DICE*SIDES-DICE+1`?
- Why are all the elements of the table `res` initialized to 0?
- What is the variable `t` storing?

Multidimensional arrays

dice_m.c

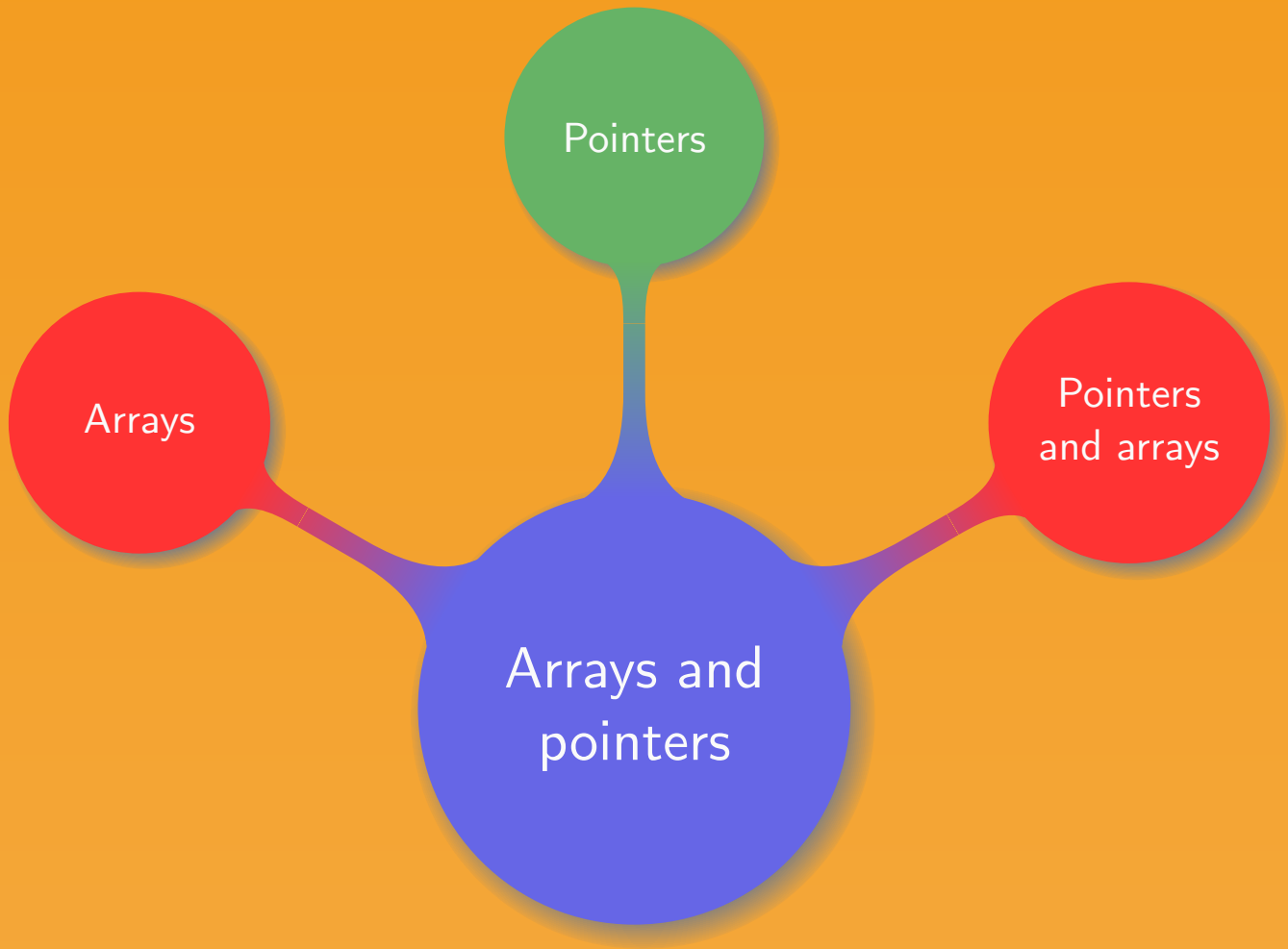
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define DICE 10
#define SIDES 6
#define ROLLS 100000
int main () {
  int i, j, t, table[DICE][ROLLS], res[DICE*SIDES-DICE+1];
  srand(time(NULL)); memset(res, 0,(DICE*SIDES-DICE+1)*sizeof(int));
  for(i=0;i<DICE;i++)
    for (j=0; j < ROLLS; j++)  table[i][j]=(rand()%SIDES)+1;
  for (i=0;i<ROLLS;i++) {
    t=0;
    for(j=0;j<DICE;j++) t+=table[j][i];
    res[t-DICE]++;
  }
  for (i=0;i<DICE*SIDES-DICE+1;i++) printf("%d (%d) ",i+DICE,res[i]);
  printf("\n");
}
```

Summary questions

In the previous three short programs:

- What three ways were used to initialize the arrays?
- Why is $i + 1$ in the first program and then $i + DICE$ in the two others printed, instead of i?
- In the multidimensional array program, is the order of the loops important? That is loop over DICE and then ROLLS vs. loop over ROLLS and then DICE.
- Rewrite the previous code (7.9) using a function taking dice, sides, and rolls as input
- Explain how multi-dimensional arrays are stored in the memory

Chapter organisation



What is a pointer?

Pointer:

- Something that directs, indicates, or points
- Low level but powerful facility available in C

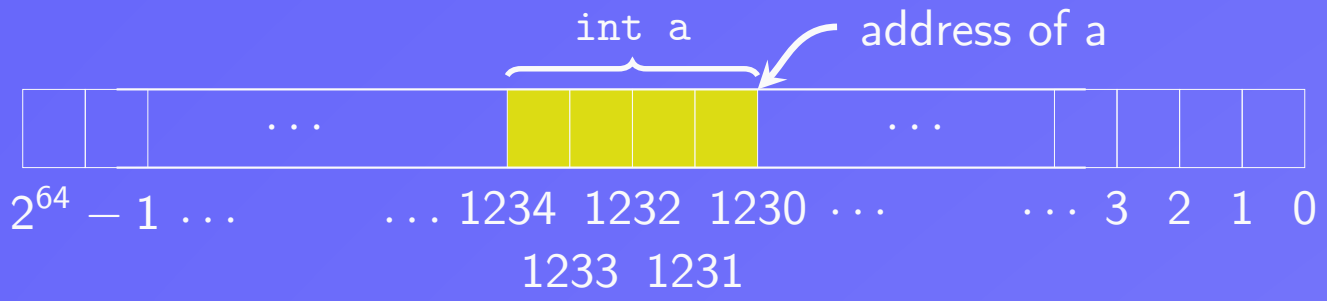
What is a pointer?

Pointer:

- Something that directs, indicates, or points
- Low level but powerful facility available in C

Pointer vs. variable:

- *Variable:* area of the memory that has been given a name
- *Pointer:* variable that stores the address of another variable



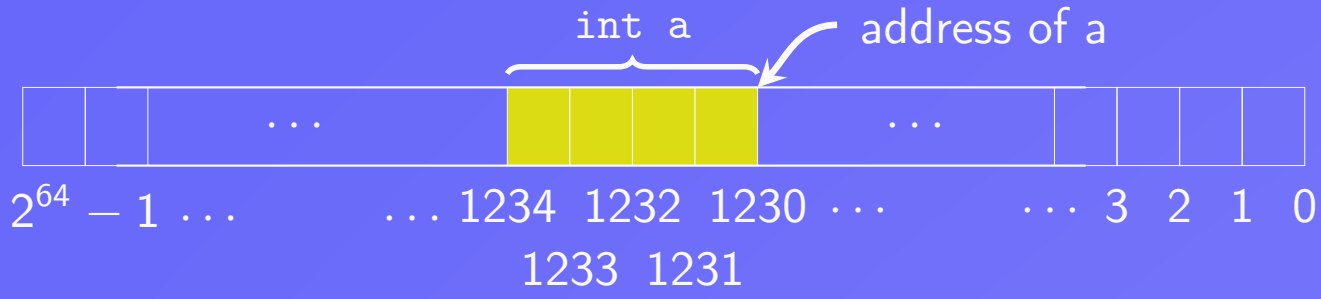
What is a pointer?

Pointer:

- Something that directs, indicates, or points
- Low level but powerful facility available in C

Pointer vs. variable:

- *Variable:* area of the memory that has been given a name
- *Pointer:* variable that stores the address of another variable



A pointer points to a variable, it is the address of the variable

How to use pointers

Handling pointers:

- The *address* of a variable `x` is `&x`
- The value stored at address `y` is `*y`
- The operator "`*`" is called *dereferencing* operator

How to use pointers

Handling pointers:

- The *address* of a variable `x` is `&x`
- The value stored at address `y` is `*y`
- The operator "`*`" is called *dereferencing* operator

Type of a pointer:

- A pointer is an address represented as a `long long int`
- It is easy to define a pointer of pointer
- The type of the variable stored at an address must be provided
- Defining a pointer: `type* variable;`

Why using pointers?

swap.c

```c
#include <stdio.h>
void swap(int a,int b);
int main() {
  int a=2, b=5;
  swap(a,b);
  printf("a = %d, ",a);
  printf("b = %d\n",b);
  return 0;
}
void swap(int a,int b) {
  int temp=a;
  a=b;
  b=temp;
}
```

swap_ptr.c

```c
#include <stdio.h>
void swap(int *a, int *b);
int main() {
  int a=2, b=5;
  swap(&a,&b);
  printf("a = %d, ",a);
  printf("b = %d\n",b);
  return 0;
}
void swap(int* a,int* b) {
  int temp=*a;
  *a=*b;
  *b=temp;
}
```

Why using pointers?

Understanding the code:

- What is the difference between the two programs?
- Which one returns the proper result?
- Why is one of the programs not working?
- Why is the other program working?
- Why were pointers used in the second program?

Example

ptr.c

```c
#include <stdio.h>
void pointers();
int main() {pointers();}
void pointers() {
  float x=0.5; float *xp1;
  float **xp2 = &xp1; xp1 = &x;
  printf("%llu %p\n%f ",xp1,&x,**xp2);
  x=**xp2+*xp1; printf("%f\n",x);
}
```

Understanding the code:

- Without running the program guess the final value of x
- Alter the program to display `*xp2`
- Explain the result

Dynamic memory

Functions to manage memory:

- Allocate *n* bytes of memory, and get a pointer on the first chunk: `malloc(n)`
- Allocate *n* blocks of size `s` each, set the memory to 0, and get a pointer on the first chunk: `calloc(n,s)`
- Adjust the size of the memory block pointed to by `ptr` to `s` bytes, and get a pointer on the first chunk: `realloc(ptr,s)`
- Frees the memory space pointed to by `ptr`: `free(ptr)`

Dynamic memory

Functions to manage memory:

- Allocate *n* bytes of memory, and get a pointer on the first chunk: `malloc(n)`
- Allocate *n* blocks of size `s` each, set the memory to 0, and get a pointer on the first chunk: `calloc(n,s)`
- Adjust the size of the memory block pointed to by `ptr` to `s` bytes, and get a pointer on the first chunk: `realloc(ptr,s)`
- Frees the memory space pointed to by `ptr`: `free(ptr)`

Any allocated memory must be released

Accessing memory

Example.

```
1  int *a=malloc(6*sizeof(int));
```

- Accessing first chunk
- Accessing the 5th chunk

Accessing memory

Example.

```
1  int *a=malloc(6*sizeof(int));
```

- Accessing first chunk

```
1  printf("%d",*a);
```

- Accessing the 5th chunk

```
1  printf("%d",*(a+4));
```

Accessing memory

Example.

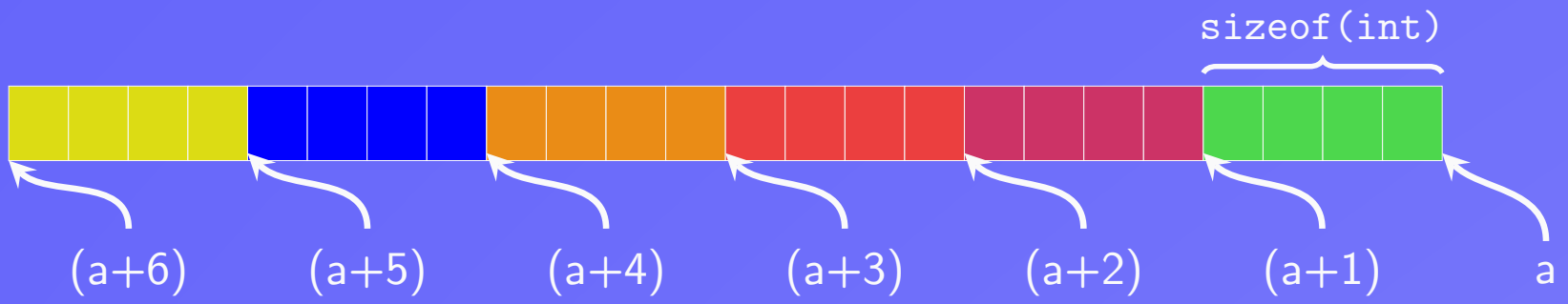
```
1  int *a=malloc(6*sizeof(int));
```

- Accessing first chunk

```
1  printf("%d",*a);
```

- Accessing the 5th chunk

```
1  printf("%d",*(a+4));
```



In this example what is `(a+6)`?

Pointers and structures

struct_ptr.c

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct person {
  char* name; int age;
} person_t;
int main () {
  struct person ya = {
    .name="Yann",
    .age=12,
  };
  person_t al={"albert",32};
  person_t* group1=malloc(3*sizeof(person_t));
  group1->name="gilbert"; group1->age=34;
  *(group1+1)=(person_t){"joseph",28};
  (*(group1+2)).name="emily"; (group1+2)->age=42;
  printf("%s %d %lu\n",ya.name, ya.age,sizeof(struct person));
  printf("%s %d %lu\n",al.name, al.age, sizeof(person_t));
  printf("%s %d\n",(group1+1)->name, (group1+2)->age);
  free(group1); return 0;
}
```

Pointers and structures

Understanding the code:

- How to use `malloc`?
- What are the different ways to access elements of a structure when the variable is not a pointer?
- What are the different ways to access elements of a structure when the variable is a pointer?
- Why should the pointer be freed at the end of the program?

General notes

Remarks on pointers:

- Not possible to choose the address, e.g. `int *p; p=12345;`
- The `NULL` pointer "points nowhere"
- An uninitialized pointer "points anywhere", e.g. `float *a;`

General notes

Remarks on pointers:

- Not possible to choose the address, e.g. `int *p; p=12345;`
- The `NULL` pointer "points nowhere"
- An uninitialized pointer "points anywhere", e.g. `float *a;`

A good practice consists in checking the memory allocation:

```
1 har* p = malloc(100);
2 if (p == NULL) {
3   fprintf(stderr, "Error: out of memory");
4   exit(1);
5 }
```

Chapter organisation

Pointer vs. array

An array contains *elements and a pointer* points *to them*

arr_ptr.c

```c
#include <stdio.h>
#include <stdlib.h>
void ptr_vs_arr();
int main () {
  ptr_vs_arr();
}
void ptr_vs_arr(){
  int a[3]={0,1,2};
  int* p=malloc(3*sizeof(int));
  *p=3; *(p+1)=4; *(p+2)=5; printf("%d %d\n",a[0], *p);
  a[0]=42;  p=a; p++ ; *p=a[2];
//a=p; p=c; p=a[0]; p=&a; a++;
  printf("%d %d %lu %lu\n",a[0], *p,sizeof(a), sizeof(p));
//  free(p);
}
```

Pointers and strings

A pointer to `char` *is different from an array of* `char`

string_ptr.c

```c
#include <stdio.h>
void str_ptr();
int main () {
  str_ptr();
}
void str_ptr(){
  char a[]="good morning!";
  char* p="Good morning!";
  printf("%c %c\n",a[0], *p);
  a[0]='t'; //*p='t';
  p=a;//a=p; p=c; p=a[0]; p=&a;
  p++; //a++;
  printf("%c %c %lu %lu\n",a[0], *p,sizeof(a), sizeof(p));
}
```

Arrays as pointers

Exercise.

Create an array `a` containing the four elements 1, 2, 3, and 4, then print `&a[i]`, `(a+i)`, `a[i]`, and `*(a+i)`

Arrays as pointers

Exercise.

Create an array `a` containing the four elements 1, 2, 3, and 4, then print `&a[i]`, `(a+i)`, `a[i]`, and `*(a+i)`

arr_ptr2.c

```c
#include <stdio.h>
void arr_as_ptr(){
  int a[4]={1, 2, 3, 4};
  for(int i=0;i<4;i++) {
    printf("&a[%d]=%p (a+%d)=%p\n"\
        "a[%d]=%d *(a+%d)=%d\n",\
        i,&a[i],i,(a+i),i,a[i],i,*(a+i));
  }
}
int main () {arr_as_ptr();}
```

Arrays and pointers

In the three previous programs:

- List what can be done with a pointer but not with an array
- List what can be done with an array but not with a pointer
- Is it possible to read a pointer as an array?
- Is it possible to read an array as a pointer?
- What is the size of a pointer, why?
- Can a `char*` be changed?

Revisiting the dice

dice_mp.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void roll_dice(int dice, int sides, int rolls){
  int i, j, t;
  int *res=calloc((dice*sides-dice+1),sizeof(int));
  int *table=malloc(dice*rolls*sizeof(int));
  for(i=0;i<rolls;i++) {
    for (j=0; j < dice; j++) table[i*dice+j]=(rand()%sides)+1;
  }
  for (i=0;i<rolls;i++) {
    t=0; for(j=0;j<dice;j++) t+=table[i*dice+j]; res[t-dice]++;
  }
  for (i=0;i<dice*sides-dice+1;i++) printf("%d (%d) ",i+dice,res[i]);
  printf("\n"); free(table); free(res);
}
int main () {
  int dice=4, sides=6, rolls=10000000;
  srand(time(NULL)); roll_dice(dice,sides,rolls);
}
```

Revisiting the dice

Understanding the code:

- How is the array `table` handled?
- What happened in the previous version with 1000000 rolls?
- Is the same happening now, why?
- How is the program organised?
- How are `malloc` and `calloc` used?

Summary on pointers

Limitation of C:

- No limit on the number of input
- Only one output
- Output cannot be an array

Summary on pointers

Limitation of C:

- No limit on the number of input
- Only one output
- Output cannot be an array

Use pointers as input (slide 7.14)

Summary on pointers

Limitation of C:

- No limit on the number of input
- Only one output
- Output cannot be an array

Use pointers as input (slide 7.14)

Common mistakes leading to `segmentation fault`:

- Memory has not been allocated
- Memory has been freed too early
- Memory is freed twice or more times
- Memory is accessed but does not belong to the program

Key points

- What are the three information necessary to define an array?
- What are `&a` and `*a`?
- Given a pointer on a structure how to access a specific field?
- Are pointers and array the same?
- What to do with unused allocated memory?
- How to have more than one output in a function?

Thank you!