Introduction to Programming and Computational Physics

Lesson n.8

Strings
Structures
Class and Objects (C++)

Char and string

A char is a one byte integer data type:

```
char ch; //declaring a char
ch = 'h'; //the corresponding ASCII code is stored

printf("%c", ch); //the character is printed
```

h

```
printf("%d", ch); //the ASCII code is printed
```

104

An array of char is called string:

```
char str[10];  str[0] = 'X'; str[1] = '6';
printf("%c", str[0]);
```

The ASCII code

0	Ctrl-@	32	Space	64	@	96	`
1	Ctrl-A	33	!	65	A	97	a
2	Ctrl-B	34	"	66	B	98	b
3	Ctrl-C	35	#	67	C	99	c
4	Ctrl-D	36	$	68	D	100	d
5	Ctrl-E	37	%	69	E	101	e
6	Ctrl-F	38	&	70	F	102	f
7	Ctrl-G	39	'	71	G	103	g
8	Backspace	40	(	72	H	104	h
9	Tab	41	)	73	I	105	i
10	Ctrl-J	42	*	74	J	106	j
11	Ctrl-K	43	+	75	K	107	k
12	Ctrl-L	44	,	76	L	108	l
13	Return	45	-	77	M	109	m
14	Ctrl-N	46	.	78	N	110	n
15	Ctrl-O	47	/	79	O	111	o
16	Ctrl-P	48	0	80	P	112	p
17	Ctrl-Q	49	1	81	Q	113	q
18	Ctrl-R	50	2	82	R	114	r
19	Ctrl-S	51	3	83	S	115	s
20	Ctrl-T	52	4	84	T	116	t
21	Ctrl-U	53	5	85	U	117	u
22	Ctrl-V	54	6	86	V	118	v
23	Ctrl-W	55	7	87	W	119	w
24	Ctrl-X	56	8	88	X	120	x
25	Ctrl-Y	57	9	89	Y	121	y
26	Ctrl-Z	58	:	90	Z	122	z
27	Escape	59	;	91	[	123	{
28	Ctrl-\	60	<	92	\	124	|
29	Ctrl-]	61	=	93	]	125	}
30	Ctrl-^	62	>	94	^	126	~
31	Ctrl-_	63	?	95	_	127	Delete

String

If we define:

```
char stat[] = "hallo world";
```

The dimension of the array will be given by the number of characters plus one, the null char \0 (it corresponds to the value 0 of ASCII code).

The two statements:

```
char d = 'r'; //a char
```

```
char d[] = "r"; //a string made of 2 char (r and \0)
```

are not equivalent

A fast way to print a string is using printf as:

```
printf(str);
```

String printing

```
#include <stdio.h>

int main()
{
    char ch[] = "C++ is a programming language.\n";
    int i=0;
    while (ch[i]!='\0')
    {
        printf("%c",ch[i]);
        i++;
    }
    return 0;
}
```

printf and sprintf

The function printf takes a string as input (and prints it to the screen):

```
printf("hallo world\n");
```

is equivalent to:

```
char stat[] = "hallo world\n";

printf(stat);
```

If the content of the string is not specified when we declare the array we have to specify its dimension and then use the sprintf function:

```
char stat[20];

sprintf(stat,"hallo world\n");

printf(stat);
```

usage of sprintf

```c
#include <stdio.h>

int main()
{
    char stat[100];
    int day = 20;
    char month[] = "April";
    int year = 2010;
    sprintf(stat, "Hallo, today is %d %s %d.\n", day,month,year);
    printf(stat);
    return 0;
}
```

A summary of all input/output commands

```
int a;
char string[100];
FILE *filepointer;
```

Input:

```
scanf("%d",&a);                    // reads from the keyboard

fscanf(filepointer,"%d",&a);       // reads from a file

sscanf(string,"%d",&a);            // reads from a string
```

Output:

```
printf("%d\n",a);                  // prints on the screen

fprintf(filepointer,"%d\n",a);     // prints into a file

sprintf(string,"%d\n",a);          // prints into a string
```

Structures

A structure is a *customizable* data type formed by a collection of variables of different types. The content of a structure is defined by the user.

```
struct nameStruct
{
   typeMember1 nameMember1;
   typeMember2 nameMember2;
   ...
   typeMemberN nameMemberN;
};
```

```
struct nameStruct nameVar;
```

Declaration of a structure of nameStruct data type

Access to the elements of the structure

```
struct date{
  int day;
  char *month;
  int year;
};
```

Once that a structure is declared, we can access its members using the expression: `structName.structMember`

```
struct date today;
today.day = 20;
today.month = "April";
today.year = 2010;
```

Structures as I/O parameters

```
struct point{
    float x;
    float y;
}
```

```
struct point Middle(struct point p1, struct point p2){
  struct point middle;
  middle.x = (p1.x + p2.x)/2.;
  middle.y = (p1.y + p2.y)/2.;
  return middle;
}
```

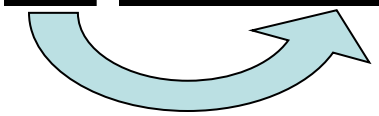
Line crossing 2 points

```c
#include<stdio.h>

struct point{
  float x;
  float y;
};

struct line{ //y=Ax+B
  float A;
  float B;
};

struct line CalcAB(struct point, struct point);

int main(){
  struct point p1,p2;
  struct line l12;
  printf("Enter the first point   x = ");
  scanf("%f",&(p1.x));
  printf("                       y = ");
  scanf("%f",&(p1.y));
  printf("Enter the second point  x = ");
  scanf("%f",&(p2.x));
  printf("                       y = ");
  scanf("%f",&(p2.y));
  if((p1.x==p2.x)&&(p1.y==p2.y)){
    printf("it's the same point...\n");
    return 0;
  }
  l12 = CalcAB(p1,p2);
  printf("The line crossing p1 and p2 is y=%fx+%f\n",l12.A,l12.B);
  return 0;
}

struct line CalcAB(struct point p1, struct point p2){
  struct line l12;
  l12.A = (p2.y-p1.y)/(p2.x-p1.x);
  l12.B = p1.y - l12.A*p1.x;
  return l12;
}
```

The `typedef` instruction

The C gives the possibility to define an "alias" for the types with the `typedef` command. For instance, if you define:

```
typedef int integernumber;
```



The following declarations are equivalent:

```
int a;

integernumber a;
```

where `int` is the usual type for integer numbers and `integernumber` is a new type name.

Sometimes it is very useful to customize your program and make it more easy to read.

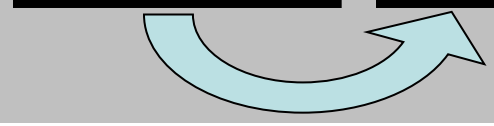
typedef and the structures

typedef is very useful with the structures because it make it more easy to define it. The following three declarations are completely equivalent:

```
struct point{
  float x;
  float y;
};

struct point A;
```

The struct point structure is defined.
The variable A is allocated as struct point type.

```
struct point{
  float x;
  float y;
};
typedef struct point point;

point A;
```



The struct point structure is defined, then the struct point type is redefined as point type.
The variable A is allocated as point type.

```
typedef struct {
  float x;
  float y;
} point;

point A;
```

The point type is immediately defined.
The variable A is allocated as point type.

Line crossing 2 points

```c
#include<stdio.h>

typedef struct {
  float x;
  float y;
} point;

typedef struct { //y=Ax+B
  float A;
  float B;
} line;

line CalcAB(point, point);

int main(){
  point p1,p2;
  line l12;
  printf("Enter the first point   x = ");
  scanf("%f",&(p1.x));
  printf("                        y = ");
  scanf("%f",&(p1.y));
  printf("Enter the second point  x = ");
  scanf("%f",&(p2.x));
  printf("                        y = ");
  scanf("%f",&(p2.y));
  if((p1.x==p2.x)&&(p1.y==p2.y)){
    printf("it's the same point...\n");
    return 0;
  }
  l12 = CalcAB(p1,p2);
  printf("The line crossing p1 and p2 is y=%fx+%f\n",l12.A,l12.B);
  return 0;
}

line CalcAB(point p1, point p2){
  line l12;
  l12.A = (p2.y-p1.y)/(p2.x-p1.x);
  l12.B = p1.y - l12.A*p1.x;
  return l12;
}
```

Array of structures

An array of structure can be defined in the following way:

```
typedef struct{
  float x;
  float y;
} point;
point p[10];
```

The following loop initializes all the elements to zero:

```
int i;
for (i=0;i<10;i++){
  p[i].x = 0.;
  p[i].y = 0.;
}
```

Pointers to structures

As for any other data type, it is possible to define pointers to structures.

```
typedef struct{
  float x;
  float y;
} point;
point p1, p2, *pp1, *pp2;
pp1 = &p1;
```

And we may access the elements of p1 using the pointer:

```
(*pp1).x = 5.3;  (*pp1).y = -3.4;
```

Since the usage of pointers to structure is widely used in C language a new operator is introduced to make the procedure faster:

```
pp1->x = 5.3;  pp2->y = -3.4;
```

Object-Oriented programming

The Object Oriented (OO) is a type of programming in which programmers define not only the data type of a data structure but also the types of operations (functions) that can be applied to the data structure.

An object is a discrete bundle of functions and procedures, all relating to a particular real-world concept, such as a bank account holder or hockey player. Other pieces of software can access the object only by calling its functions and procedures that have been allowed to be called by outsides.

Programmers can create relationships between one object and other. For example, objects can *inherit* many of its features from existing objects.

To perform object-oriented programming, one needs an object-oriented programming language. Java, C++ are two of the more popular languages

Classes and objects

```
class Point
{
   int _x, _y;                  // point coordinates (private)
   public:                      // begin interface section
   void setX(const int val);
   void setY(const int val);
   int getX() { return _x; }
   int getY() { return _y; }
};
Point apoint;    //declare an object (instance of the class Point)
```

Elements of classes are *private* by default (not accessible from outside of the class)

Methods are not *global*, they are defined inside the classes

Constructors

Constructors are methods which are used to initialize an object at its definition time.

```
class Point
{
   int _x, _y;
   public:
   Point() {_x = _y = 0; }          //constructor
   void setX(const int val);
   void setY(const int val);
   int getX() { return _x; }
   int getY() { return _y; }
};
```

Constructors

Constructors have the same name of the class (thus they are identified to be constructors). They have no return value. As other methods, they can take arguments.

```
class Point
{
    int _x, _y;
    public:
    Point() { _x = _y = 0; }
    Point(const int x, const int y) { _x = x; _y = y; }
    void setX(const int val);
    void setY(const int val);
    int getX() { return _x; }
    int getY() { return _y; }
};
```

} Function overloading

Constructors are implicitly called when we define objects of their classes:

```
Point apoint;              // Point::Point()
Point bpoint(12, 34);      // Point::Point(const int, const int)
```

Destructors

Destructor is a method called once to destroy the object

```
class Point
 {
   int _x, _y;
   public:
   Point() { _x = _y = 0; }
   Point(const int x, const int y) { _x = xval; _y = yval; }
   ~Point() { /* Nothing to do! */ }     //destructor
   void setX(const int val);
   void setY(const int val);
   int getX() { return _x; }
   int getY() { return _y; }
};
```

Destructors take no arguments.