Information Processing 2 — Week 11 exercises

Complete the first five exercises in-class on your own laptop. Ask an instructor to check your work during the class, as soon as you finish answering each question. A **late penalty of 50%** will be applied to answers checked after class.

Complete the remaining exercises in-class if possible, or before the start of the next exercise class at latest. Exercises checked **more than one week** after the class will have a **50% late penalty** applied.

1 A structure for Cartesian points

The lecture slides show a structure type for storing 2D Cartesian points called `struct point` that has two `int` members called `x` and `y`.

```
struct point
{
    int x, y;
};
```

Modify the member type declarations for `struct point` so that both members have type `double` instead of `int`. Write a function `struct point point_newXY(double x, double y)` that creates a point and returns it.

Check your definitions using the following program (download '`11-01-point.c`' from Teams):

```
int main(int argc, char **argv)
{
    struct point p = point_newXY(3.0, 4.0);
    printf("%f,%f", p.x, p.y);
    putchar('\n');
    return 0;
}
```

Verify that the output is: `3.000000,4.000000`

▷▷ Ask an instructor to check and record your work.

2 Using functions to access properties of points

Your `main` program accesses the internal structure of `p` directly, to obtain its `x` and `y` member values. A more flexible way to access those values is to use a pair of functions:

`double point_x(struct point p)` returns the `x` value stored in `p`
`double point_y(struct point p)` returns the `y` value stored in `p`

Write these two functions and then modify your call to `printf()` so that it uses them to obtain the values of the `x` and `y` members of `p`.

▷▷ Ask an instructor to check and record your work.

3 A utility function to print a point

Your programs might need to print many points.

Write a function `struct point point_printXY(struct point p)` that prints `p` in the same format as above and then returns `p`. In your `main` function, replace your call to `printf()` with '`point_printXY(p);`'. The output from your program should be unchanged.

▷▷ Ask an instructor to check and record your work.

4 Synthesising additional properties of points

You already wrote two functions, `point_x()` and `point_y()`, that return the `x` and `y` 'properties' of a point. These properties were stored directly inside the point. Functions that return other properties of points, that are not stored directly, can be implemented by computing those property value. For example, polar points are stored as an *angle* (anti-clockwise from the X axis) and a *radius* (distance from the origin).

Write a function `double point_r(struct point p)` that returns the *radius* (distance from the origin) of point `p`.

Hints:

- You might need the function `double sqrt(double)` from the <`math.h`> library.
- On some operating systems when you compile a program that uses the <`math.h`> library you will have to add '`-lm`' to the end of your compile command, like this:

```
cc -o prog prog.c -lm
```

Test your function with the following program (download '`11-04-point_r.c`' from Teams):

```
int main(int argc, char **argv)
{
    struct point p = point_newXY(3, 4);
    point_printXY(p);
    putchar('\n');
    printf("%f\n", point_r(p));

    return 0;
}
```

Verify that the value printed for the radius is: `5.0`

▷▷ Ask an instructor to check and record your work.

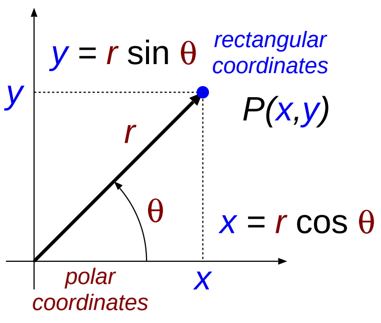
5 Creating points from polar coordinates

Sometimes it is convenient to create a point using its polar coordinates.

Write a new function called `struct point point_newRA(double r, double a)` that creates a point from its radius `r` and angle `a` (measured anti-clockwise from the positive X axis, in *degrees*).

Hints:

- The relationship between XY (Cartesian) and polar points is shown in the diagram on the right.
- <`math.h`> declares the functions `double sin(double)` and `double cos(double)`. The same header file defines `M_PI` as a `double` whose value is π.
- The `sin()` and `cos()` functions work in radians, not degrees, so you will have to convert as follows: divide the angle in degrees by `360.0` and then multiply by `2.0*M_PI` to obtain the angle in radians.



Note: On some operating systems when you compile a program that uses the <`math.h`> library you will have to add '`-lm`' to the end of your compile command, like this:

```
cc -o prog prog.c -lm
```

Check your function with the following program:

```
int main(int argc, char **argv)
{
    struct point p = point_newRA(5, 53.13);
    point_printXY(p);
    putchar('\n');
    return 0;
}
```

Verify that the value printed is almost `3.0,4.0`.

▷▷ Ask an instructor to check and record your work.

6 A structure for managing strings

Define a new type `struct string` that contains two members: `char *chars` is a pointer to some memory for storing a string value, and `int size` is the number of characters currently stored in that memory.

Check your type definition using the following program (download '`11-06-string.c`' from Teams):

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

/* define struct string here */

int main(int argc, char **argv)
{
    struct string s = { strdup("hello"), 5 };
    printf("\"%.*s\"\n", s.size, s.chars);
    return 0;
}
```

▷▷ Ask an instructor to check and record your work.

7 Creating and printing strings

Write the function `struct string string_new(char *s)` that creates a new string structure. Initialise the `chars` member with a copy of `s` obtained using `strdup()`, as in the previous exercise. Calculate the initial value for the `size` member by calling `strlen(s)` which returns the length of `s` excluding the final 'nul' character.

Write the function `struct string string_println(struct string s)` that prints the contents of the string `s` followed by a newline character, as in the previous exercise. Return `s` as the result.

Check your functions using the following program:

```
int main(int argc, char **argv)
{
    struct string s = string_new("hello");
    string_println(s);
    return 0;
}
```

▷▷ Ask an instructor to check and record your work.

8 Using pointers to structures as function parameters

Structures are usually not passed to functions by copy, as this would be inefficient for large structures with many members. Instead most structures are passed to functions *by reference*. The argument is a pointer to the structure, not a copy of the structure. This also allows the function to modify the contents of the caller's structures, when necessary.

Modify your `string_println()` function so that its parameter (and result) is a pointer to the string structure, instead of a copy of the structure. Use the '`&`' operator to obtain the address of the structure.

Check your function using the following program:

```
int main(int argc, char **argv)
{
    struct string s = string_new("hello");
    string_println(&s);
    return 0;
}
```

▷▷ Ask an instructor to check and record your work.

9 Appending a character to a string

Write a function `struct string *string_append(struct string *s, int c)` that appends the character `c` to the end of the string `s`. Hints:

- The function `realloc(void *p, size_t n)` changes the amount of memory available at the address `p` (which must have been obtained with `malloc()`, `calloc()`, `realloc()`, or `strdup()`) to be at least `n` bytes. To make space for one additional character in `s` you can therefore use:

```
        s->chars = realloc(s->chars, s->size + 1);
```

- Don't forget to increase `s->size` by 1 after appending the character.

Check your function works using this program:

```
int main(int argc, char **argv)
{
    struct string s = string_new("hello");
    string_append(&s, '!');
    string_println(&s);
    return 0;
}
```

Verify that the program prints: `"hello!"`

▷▷ Ask an instructor to check and record your work.

10 Concatenating two strings

Structures and functions can work together to make powerful abstractions for the programmer. In our string structure we can hide a lot of the memory management needed when manipulating strings.

Write a function

```
struct string *string_extend(struct string *s, struct string *t)
```

that appends all the characters in string `t` to the end of string `s`, and then returns `s`.

Hints:

- You can append an entire string `t` to another string `s` by calling `string_append(s, c)` once for each character `c` in the string `t`.
- Another (better) way to do it is to call `realloc()` just once, to grow `s->chars` by `t->size` characters, and then copy the contents of `t->chars` to the end of the original contents of `s` at the address `s->chars + s->size`. This copy can be performed efficiently using the function

  ```
      memcpy(void *destination, void *source, size_t n)
  ```

  (declared in `<stdlib.h>`) which copies `n` bytes from `source` to `destination`. (Use whichever method you are most comfortable with, but using the second method will definitely impress your instructors.)

Check that your function works with the following program:

```
int main(int argc, char **argv)
{
    struct string s = string_new("hello");
    struct string t = string_new("world");
    string_append(&s, ' ');
    string_extend(&s, &t);
    string_println(&s);
    return 0;
}
```

▷▷ Ask an instructor to check and record your work.

Challenges [1 bonus point]

Complete both challenges to earn a bonus point.

11 The angle of a point

Add a function `double point_a(struct point p)` to your point program which returns the angle of the point `p` (measured anti-clockwise from the positive X axis).

Add another function `point_printRA(struct point p)` which prints `p` as its radius and angle.

With these two additions you can now switch easily between working in polar coordinates or in rectangular coordinates, even with the same point(s).

Check your function by creating several points using polar coordinates and then printing their angles, which should be the same as the angle used to create them.

Hint: the `<math.h>` library has a function `double atan2(double y, double x)` which you might find useful.

12 Rectangles

The lecture slides show how to place two `struct point` values inside a `struct rectangle` type to define the `origin` and `corner` of a rectangle.

Implement the `struct rectangle` type, a constructor function

```
struct rectangle rectangle_new(struct point origin, struct point corner)
```

and a function to print rectangles:

```
struct rectangle *rectangle_println(struct rectangle *r)
```

Then implement a function `int rectangle_includes(struct point *p)` that returns 1 if `p` is inside the rectangle `r` (inclusive of the origin point but exclusive of the corner point) and 0 if `p` is outside `r`.