Sesión 06: Punteros

Hoja de problemas

Programación para Sistemas

Ángel Herranz
aherranz@fi.upm.es
Universidad Politécnica de Madrid

2022-2023

Ejercicio 1. Repasa las transparencias de clase. Las transparencias de este tema están repletas de pruebas que tienes que entender en detalle. La memoria dinámica, especialmente cuando se viene de lenguajes con recolección automática de basura, se hace muy complicado, mucho más al combinarlo con la estrecha relación entre arrays y punteros de C. Entender cada transparencia se hace esencial.

Ejercicio 2. Poco a poco vamos viendo nueva sintaxis para las expresiones del lenguaje C. En esta sesión hemos descubierto dos sintaxis nuevas y muy importantes:

$$*e \qquad \&e$$

Antes de continuar vuelve a asegurarte de que entiendes la semántica de ambas sintaxis:

- $*e$: se evalúa e, su resultado es entonces interpretado como una dirección de memoria, y $*e$ hace referencia al contenido de dicha dirección de memoria.
- $\&e$: en este caso e no puede ser cualquier expresión, sólo puede ser un *lvalue*[1], normalmente un identificador. El significado es «la dirección de memoria de e».

Ejercicio 3. No olvides realizar una implementación correcta de la función `intercambiar`.

Ejercicio 4. Ejecuta la función `intercambiar` bajo el control de `gdb` y vete explorando las variables.

[1] Todo aquello que puede aparecer en la izquierda de una asignación.

Ejercicio 5.

RPN (*Reverse Polish Notation*)

> The following algorithm evaluates postfix expressions using a stack, with the expression processed from left to right:
>
> ```
> for each token in the postfix expression:
>   if token is an operator:
>     operand_2 = pop from the stack
>     operand_1 = pop from the stack
>     result = evaluate token with operand_1 and operand_2
>     push result back onto the stack
>   else if token is an operand:
>     push token onto the stack
> result = pop from the stack
> ```
>
> Reverse Polish notation (Wikipedia)

En este ejercicio tendrás que programar una calculadora *polaca inversa*:

- Los *tokens* serán floats, operadores (`'+'`, `'-'`, `'*'`, `'/'`) y el caracter de *fin de expresión* (`'='`)
- Leemos la expresión de la entrada estándar con `scanf`[2]:

  ```
  float operando;
  char operador[2];
  scanf("%f", &operando);
  scanf("%s", operador);
  ```

- Utilizaremos un array para implementar la pila de operandos
- Asumiremos que la pila no puede crecer en más de 1000 elementos

Así deberá comportarse tu programa:

```
$ ./rpn
15 7 1 1 + - / 3 * 2 1 1 + + - =
5
$ |
```

Ejercicio 6. Si no lo has hecho ya, modifica tu programa `rpn` para acceder a la pila de operandos utilizando punteros.

Ejercicio 7. Transcribe el siguiente programa `nodina.c`.

[2] Utiliza el manual (`man 3 scanf`), o mejor el libro K&R, apéndice B, sección1.3.

```c
#include <stdio.h>

int main() {
  int n;
  scanf("%d",&n);
  int a[n];
  printf("%p\n",a);
  return 0;
}
```

Como ves, el programador pretende que el usuario introduzca un número para luego *crear* un array con tantos elementos como los indicados por el usuario. Compílalo con

```
cc -Wall -Werror -g -ansi -pedantic -o nodina nodina.c.
```

- ¿Entiendes el mensaje del compilador?
- ¿Cómo debes hacerlo para ser *puro C*?
- ¿Qué ocurre si eliminas el *flag* `-pedantic`?

Ejercicio 8. Consultar el manual `man 3 malloc`. En dicha página del manual encontrarás la documentación de varias operaciones funciones:

- `void *malloc(size_t size);`
- `void free(void *ptr);`
- `void *calloc(size_t nmemb, size_t size);`
- `void *realloc(void *ptr, size_t size);`
- `void *reallocarray(void *ptr, size_t nmemb, size_t size);`

```
MALLOC(3)              Linux Programmer's Manual             MALLOC(3)

NAME
       malloc,  free, calloc, realloc - allocate and free dynamic mem-
       ory

SYNOPSIS
       #include <stdlib.h>

       void *malloc(size_t size);
       void free(void *ptr);
       void *calloc(size_t nmemb, size_t size);
       void *realloc(void *ptr, size_t size);
       void *reallocarray(void *ptr, size_t nmemb, size_t size);

   Feature   Test   Macro   Requirements   for   glibc    (see    fea-
```

```
   ture_test_macros(7)):

       reallocarray():
           _GNU_SOURCE

DESCRIPTION
       The  malloc()  function  allocates  size  bytes  and  returns a
       pointer to the allocated memory.  The memory  is  not  initial-
       ized.   If  size  is 0, then malloc() returns either NULL, or a
       unique pointer value that can later be successfully  passed  to
       free().

...
```

Ejercicio 9. La llamada a la función `calloc(`n`, `s`)` de la biblioteca estándar devuelve un puntero a n objetos de tamaño s cada uno con toda la memoria inicializada a 0. En este ejercicio tendrás que implementar tú mismo la función `calloc`, puedes llamarla `micalloc ;)`, llamando a la función `malloc`.

Ejercicio 8-6 del K&R

⏭ **Ejercicio 10.** Para poder probar tu implementación de ordenación de enteros propuesta en las transparencias, te sugerimos usar la potencia de Bash.

Las siguientes órdenes de Bash generan un fichero `enteros.txt` con una entrada aleatoria apropiada para tu programa:

```
R=$RANDOM
echo $R > enteros.txt
for ((i = 0 ; i < $R ; i++)); do
  echo $RANDOM >> enteros.txt;
done
```

A mi me ha salido esto:

```
$ cat enteros.txt
5
29022
957
13682
10575
22773
$ |
```

Ahora, si has llamado a tu programa `ordenar_enteros`, puedes ejecutarlo así:

```
$ ./ordenar_enteros < enteros.txt
957
10575
13682
22773
29022
$ |
```

💻 **Ejercicio 11.** Modifica el programa que ordena ristras de enteros (versión final de las transparencias) para que la ordenación la realice una función `bubble_sort`. Este ejercicio está muy orientado a que puedas entender los siguientes puntos:

- Como los punteros y los arrays *son lo mismo* en C, se puede declarar un array como argumento de la función. Como es un puntero, lo que modificamos es aquello a lo que apunta, que resulta ser el contenido del array.
- En la sesión anterior ya vimos que no era posible conocer la longitud de un array cuando es argumento de una función. Por ello es necesario que la función de ordenación reciba como argumento la longitud del array.

💻 **Ejercicio 12.** Modifica el programa anterior para usar un algoritmo de ordenación más rápido que *bubble sort*: *quick sort* o *merge sort* por ejemplo.