UNIVERSIDAD DE COSTA RICA

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0117 Programación Bajo Plataformas Abiertas

EIE
Escuela de
Ingeniería Eléctrica

C: Punteros Parte 2

Ing. Andrés Mora Zúñiga - andres.morazuniga@ucr.ac.cr

I Ciclo 2017

Agenda

1. Memoria dinámica
   - Stack y Heap
   - Localización y liberación de memoria
2. Punteros Dobles
   - Localización de memoria en punteros dobles
   - Liberar memoria de punteros dobles
   - Ejercicio

Memoria dinámica

Stack y Heap I

- Stack
  - Se almacena en RAM.
  - Variables creadas en stack salen de scope automáticamente y la memoria se deslocaliza de manera automática.
  - La memoria se localiza mucho mas rápido en comparación al Heap.
  - Realmente está implementada con una pila como estructura de datos.
  - Almacena variables locales, direcciones de retorno y se utiliza para pasar parámetros a funciones.
- Heap
  - Se almacena en RAM.
  - En C, las variables en heap tienen que ser liberadas de manera manual con free y nunca salen del scope.
  - Más lento de localizar en comparación al Stack.
  - Se asigna a manera de solicitud para localizar un bloque de memoria requerido por el programa.
  - Puede haber fragmentación de memoria cuando se localiza y libera mucho la memoria.

Stack y Heap II

- Stack
  - Se puede desbordar (Por ejemplo: muchas llamadas recursivas)
  - Puede ser utilizada sin necesidad de punteros.
  - Se suele utilizar cuando ya se sabe la cantidad de memoria necesaria al momento de compilación, y no de debe ser muy grande.
  - Normalmente tiene un tamaño máximo definido cuando el proceso inicia.
- Heap
  - En C, la memoria reservada en heap es accedida a través de punteros. Localizada con malloc y liberada con free.
  - La localización puede fallar si se solicitan bloques de memoria muy grandes.
  - Se acostumbra utilizar el heap si no se sabe exactamente cuanta memoria se va a necesitar para una tarea en tiempo de ejecución.
  - Es la única manera de hayan "memory leaks".

Localización y liberación de memoria

```
int main(int argc, char** argv){
        //Declaration
        int* ptr;
        //Pointer does not have
        // memory to point to
        //Allocation needed!
        ptr = malloc(10*sizeof(int));
        //...
        free(ptr);
        //free memory when is no
        //longer needed
return 0;
}
```

- Al inicio ptr no tiene memoria a la cual apuntar, por lo que no puede almacenar ningún valor.
- Luego de la llamada a malloc, el puntero `ptr` cuenta con un bloque capaz de almacenar hasta 10 `int`.
- Cuando el puntero ya tiene memoria asignada, se le puede asignar valores a esta memoria a través de la notación de puntero *ptr, *(ptr+i), o usando la notación de arreglos ptr[0], ptr[i].
- Las lecturas y las asignaciones se hacen utilizando la misma notación.

Punteros Dobles

Localización de memoria en punteros dobles

```
//Lets think double pointers
//as if they were a matrix
int** ptr;
//Allocate memory for rows (int*)
ptr = malloc(3*sizeof(int*)) ;
//Allocate memory for each
//element in columns (int)
for(int i=0;i<3;i++){
ptr[i] = malloc(6*sizeof(int));
}
int c = 0;
for(int i=0;i<3;i++){
        for(int j=0;i<6;j++){
                ptr[i][j] = c;
                c++;
        }
}
```

- Al inicio ptr no tiene memoria a la cual apuntar, por lo que no puede almacenar ningún valor.
- Luego de la llamada a malloc, el puntero `ptr` cuenta con un bloque de 3 espacios capaz de almacenar int* (punteros a enteros).
- Posteriormente, se hace un for, con tantas llamadas a malloc como filas tiene la matriz. Luego del for, si todos los mallocs funcionaron, cada uno de las filas tiene la posibilidad de almacenar 6 `int` cada una. Por lo que se tiene una matriz de 3 filas y 6 columnas.
- Finalmente se puede utilizar toda la memoria que ha sido localizada.

Liberar memoria en punteros dobles

```
//We need to make as many
//frees as they were mallocs
//Starting backwards, first
//with the columns
for(int i=0;i<3;i++){
        free(ptr[i])
}
//Finally the pointer for row
free(ptr)
```

- Al igual que en el caso de un puntero sencillo, cada una de las llamadas a malloc para asignar a "filas" o a "columnas" debe venir acompañado de una llamada a free.
- Se debe liberar primero los punteros "internos", pues si se libera primero el "externo", luego no se cuenta con los otros punteros que apuntan a los bloques de memoria "internos".

Ejercicio

- Realice una tabla modelo de memoria que ejemplifique el contenido de la memoria de la diapositiva de "Localización de memoria en punteros dobles". Puede utilizar las direcciones que desee, siempre y cuando los bloques que son continuos en memoria, realmente lo sean en el modelo también.

UNIVERSIDAD DE COSTA RICA

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0117 Programación Bajo Plataformas Abiertas

EIE
Escuela de
Ingeniería Eléctrica

C: Punteros Parte 2

Ing. Andrés Mora Zúñiga - andres.morazuniga@ucr.ac.cr

I Ciclo 2017