CSE 1320 - Intermediate Programming

Dynamic Memory Allocation

Alex Dillhoff

University of Texas at Arlington

Dynamic Memory Allocation

C gives the programmer the tools to allocate memory dynamically through several functions in `stdlib.h`.

Dynamic memory allocation is a powerful tool which allows our programs to adapt to varying inputs, but it comes with increased development overhead.

Dynamic Memory Allocation

Functions related to dynamic memory allocation:

- `malloc`
- `calloc`
- `realloc`
- `free`

malloc

```
void * malloc(size_t size);
```

Allocates `size` bytes and returns a pointer to the allocated memory. This memory is **not** initialized.

If the system cannot allocate memory, `NULL` is returned.

calloc

```
void * calloc(size_t nmemb, size_t size);
```

Allocates memory for an array of `nmemb` elements of `size` bytes each and returns a pointer to the memory.

`calloc`

`calloc()` also zero-initializes the memory reserved.

realloc

```
void * realloc(void *ptr, size_t size);
```

Changes the size of the memory block points to by `ptr` to `size` bytes. If `ptr` is `NULL`, then the call is equivalent to `malloc`.

If the new size is bigger than the previous size, the new data is **not** initialized.

free

```
void free(void *ptr);
```

Frees the memory space pointed to by `ptr`, which must have been returned by a previous call to `malloc`, `calloc`, or `realloc`.

Heap vs. Stack

So far, we have created variable that have a fixed size of data.

The memory for these variables exists on the **stack** and is allocated and deallocated by compiler instructions.

Heap vs. Stack

The **stack** has several important properties.

- Linear data structure
- High-speed access
- Memory is not fragmented
- Memory cannot be resized

Heap vs. Stack

The **heap** differs from the stack in the following ways.

- Memory is resizable using `realloc`
- Memory can be fragmented
- Developer is responsible for managing memory

Void Pointer

Memory allocation functions like `malloc` return `void *`.

This allows the returned block of memory to be reallocated to match the desired type.

Dynamic Memory Allocation

Examples

- Allocate scalar values
- Allocate 1D array
- Allocate 2D array of pointers
- 2D array as 1D memory space

Pointers and Storage

It is possible to assign a pointer to another pointer, but remember that both pointers will then refer to the same location in memory.

This could cause unintended side effects in your program.

Pointers and Storage

Example: strcpy versus assignment

Dynamic Memory and Files

Dynamic memory allocation allows the developer to work with unknown file sizes and structures.

For example, when loading raw flight data, we do not know at compile time how many entries are in the file to be loaded. Dynamic memory allocation allows our programs to adapt to these conditions.

Dynamic Memory and Files

Example: Loading flight data

Valgrind

Valgrind is a debugging and profiling tool for C and C++ programs.

One of the components, `memcheck`, is particularly useful for debugging memory leaks.

Preparing Your Program for Valgrind

When using Valgrind, it is recommended to compile your program with the following flags:

- `-g` - includes debugging information so Valgrind can report exact lines.
- `-O0` - removes optimization to improve the Valgrind's accuracy in reporting.

Using Valgrind

If your program runs like this:

```
program arg1
```

You run it using Valgrind like this:

```
valgrind --leak-check=yes program arg1
```

Using Valgrind

Example: Using Valgrind to detect errors.