CSA2090: Systems Programming

Introduction to C

Lecture 2: Arrays and Structures

Dr. Christopher Staff

Department of Computer Science & AI

University of Malta

cstaff@cs.um.edu.mt

Aims and Objectives

- Basic data types in C are `char, int, long, short, float, long double`
- C has limited support for *aggregates* of the same type (arrays)…
- … and for different types (structures)

Arrays

```
char letters[50];
char values[50][30][10];
int nums[10];
```

- C has NO subscript checking…

```
letters[55] = 'd';
letters[-3] = 'x';
thisvalue = values[50][38][9];
```

Single quotes for chars!

  - See letters.c
- In C, array subscripts start at 0!
  - letters[n] = {letters[0], letters[1] … letters[n-1]}

Structures

- Can be used as “records”
- Essentially, programmers can create new data types!

```
struct person {
    int age;
    int height;
    char surname[20];
};
```


Structures

- Can be used as “records”
- Essentially, programmers can create new data types!


```
struct person {
    int age;
    int height;
    char surname[20];
};
```


Structures

- Can be used as “records”
- Essentially, programmers can create new data types!

```
struct person {
    int age;
    int height;
    char surname[20];
};
```

Structures

- Can be used as “records”
- Essentially, programmers can create new data types!

```
struct person {
    int age;
    int height;
    char surname[20];
}fred, jane; /* to create
variables */
```


Structures and Typedefs

```
struct person fred, jane, chris;
/* creates variables fred, jane
and chris of type struct person*/
```

- Unlike arrays, structures can be passed to and returned from functions

Structures and Typedefs

- Can create a new data type using `typedef`

```
typedef struct {
   int age;
   int height;
   char surname[20];
 } person;
 // creates new data type person
```


Structures and Typedefs

```
person fred, jane, chris; /*
creates three variables of
type person */
```

- `struct` and `typedef` create new variable *types* but do not create new variables!
  - No *memory* set aside
  - Memory set aside when variables are defined and their type is declared

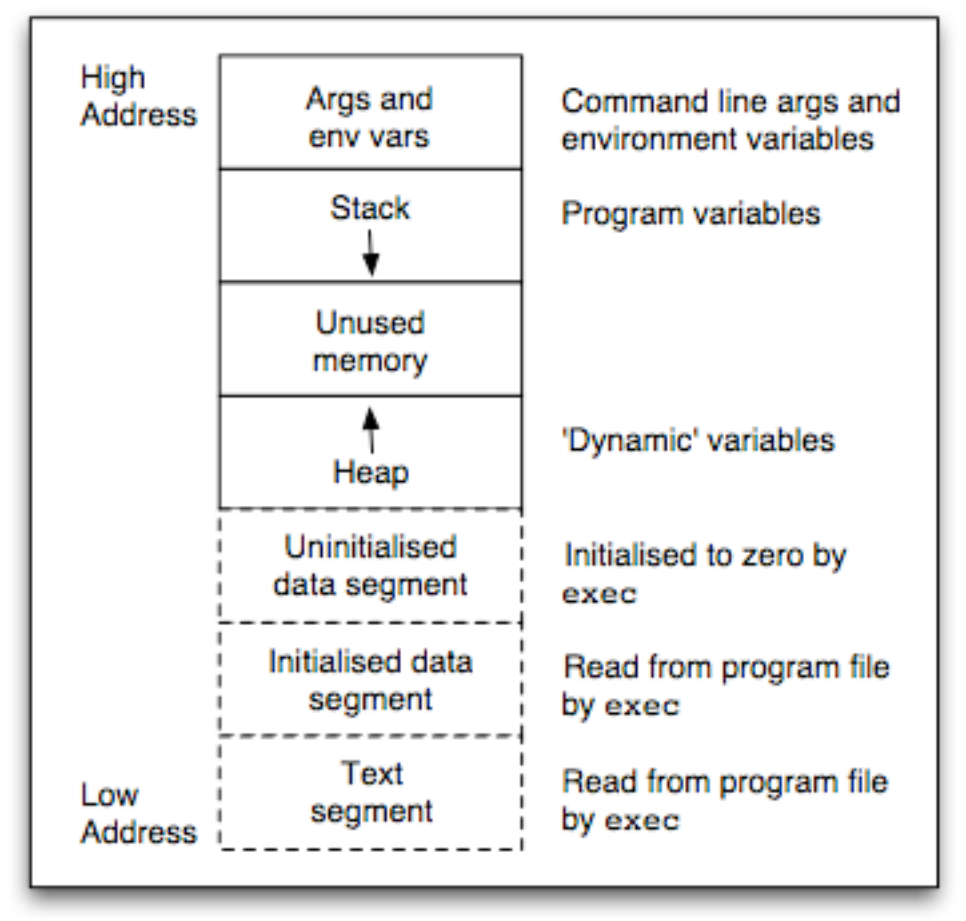
Accessing structure members

- `fred.age = 50;`
- `currHeight = jane.height;`
- Can copy entire structures
  - `fred = jane;`
- But cannot *compare* structures using ==
  - Must compare field by field, or write, e.g., comparePerson function

sizeof

- The `sizeof` operator can be used to check the size of structures, arrays, and variables
- See the sizeof.c program

C and memory: Virtual Memory



‘Static’ variables

- Some variables are *defined* (e.g., `int x`)
- C calculates how much space is required at compilation time
- All ‘static’ space requirements are reserved in the stack
- We’ll discuss the heap when we cover dynamic memory management

Variables and Memory

- When a variable is created, space is reserved in memory (in the stack)