EECS 22: Advanced C Programming

Lecture 9

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 9: Overview

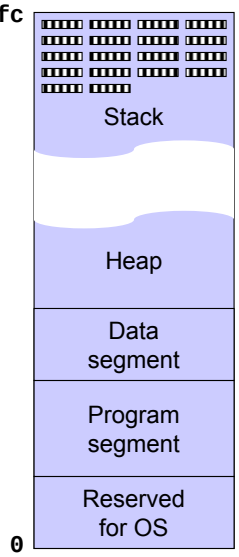
- Review Variable Lifetimes
- Memory Organization
  - Memory segmentation
  - Memory errors
- Storage Classes
  - Program example **`StorageClasses.c`**
- Recursion Revisited
  - Program example **`Fibonacci.c`**
  - Program example **`Fibonacci2.c`**

Variable Lifetimes

- Lifetime of Variables
  - Global variables (storage class **`static`**, **`extern`**)
    - From program start to end
  - Local variables (storage class **`register`**, **`auto`**)
    - From beginning of execution of their compound statement
      - Stack frame entry
    - To leaving their compound statement
      - Stack frame exit
  - Function parameters (storage class **`register`**, **`auto`**)
    - From beginning of function call
    - To returning from the function call
  - Dynamically allocated objects (more details in Lecture 12)
    - From successful return of **`malloc()`**
    - To call of **`free()`**

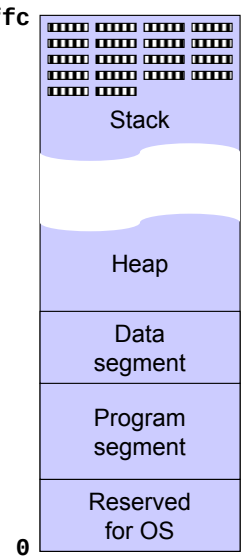
Memory Organization

- Memory Segmentation
  - typical (virtual) memory layout on processor with 4-byte words and 4 GB of memory
  - Stack
    - grows and shrinks dynamically (from top)
    - contains function call hierarchy
    - stores stack frames with local variables
  - Heap
    - “free” storage
    - dynamic allocation by the program
  - Data segment
    - global (and **`static`**) variables
  - Program segment (aka. text segment)
    - program instructions (binary code)
  - Reserved area for operating system



Memory Organization

- Memory Segmentation
  - typical (virtual) memory layout on processor with 4-byte words and 4 GB of memory
- Memory errors
  - *Out of memory*
    - Stack and heap collide
  - *Segmentation fault*
    - access outside allocated segments
    - e.g. access to segment reserved for OS
  - *Bus error*
    - mis-aligned word access
    - e.g. word access to an address that is not divisible by 4



Storage Classes

- C Language distinguishes 2 Storage Classes
  - (but uses 5 keywords and a default, depending on scope)
  - Automatic (i.e. on the stack)
    - **`auto`** local variable, on stack (default)
    - **`register`** local variable, in register (preferred) or on stack
  - Static (i.e. in the data segment)
    - **`static`** static variable in data segment
    - **`extern`** declaration of global variable in data segment
  - At compile-time, a 3rd "storage class" exists
    - **`typedef`** definition of an alias for a type at compile time (no storage)

Storage Classes

Keyword	Global Scope	Local Scope
(none)	Global variable in data segment	Local variable on stack
`auto`	n/a	Local variable on stack
`register`	n/a	Local variable on stack or in register (preferred)
`static`	Global variable in data segment (int. linkage)	Local variable in data segment
`extern`	Declaration of global variable in data segment (ext. linkage)	Declaration of global variable in data segment (ext. linkage)
`typedef`	Alias for a type at compile time (no storage in memory)	Alias for a type at compile time (no storage in memory)

Storage Classes vs. Linkage

- ANSI C language uses the keywords **`extern`** and **`static`** not only for the specification of storage classes, but also for specification of *linkage* of functions and variables
  - Unfortunate re-use of keywords easily causes confusion!
- Linkage
  - External Linkage: **`extern`**
    - The object is accessible across multiple translation units
    - Example:
```
extern double sqrt(double x);
extern const double pi;
```
  - Internal Linkage: **`static`**
    - The object is accessible in its own translation unit only
    - Example:
```
static double my_sqrt(double x);
static const double my_pi = 3.1415927;
```
- We will introduce programs with multiple translation units in detail in the following lectures

Storage Classes

- Program example: **`StorageClasses.c`** (part 1/3)

```
/* StorageClasses.c: example for storage classes and linkage */
/* author: Rainer Doemer                                      */
/*                                                            */
/* modifications:                                             */
/* 10/13/13 RD  initial version                               */

/*** global scope ***/

       void f(int); /* global function (defined below) */
extern void g(int); /* global function (defined somewhere else)*/
static void h(int); /* internal function (defined below) */

        double x;   /* global variable (defined here) */
extern  double y;   /* global variable (defined somewhere else)*/
static  double z;   /* internal global variable (defined here) */

typedef double t;   /* type definition */

...
```


Storage Classes

- Program example: **`StorageClasses.c`** (part 2/3)

```
...
void f(int p)
{
    /*** local scope ***/

            int i; /* local variable (on stack) */
    auto    int j; /* local variable (on stack) */
    register int r; /* local variable, preferably in register */
    static   int n = 0; /* static local variable */

    n++; /* count executions of this function */
    for(i=0; i<n; i++)
    { for(j=0; j<p; j++)
      { g(i*j);
      }
    }
    for(r=0; r<1000000; r++)
    { h(r);
    }
}
...
```

Storage Classes

- Program example: **`StorageClasses.c`** (part 3/3)

```
...

static void h(int p)
{
    g(p + (x*y*z));
}

/* EOF */
```


Recursion Revisited

- Example: Fibonacci series
  - Mathematical properties:
    - The first two numbers are 0 and 1
    - Every subsequent number is the sum of the previous two
  - Recursive definition:
    - Base case: $fibonacci(0) = 0,\ fibonacci(1) = 1$
    - Recursion step: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$

Recursion Revisited

- Program example: **`Fibonacci.c`** (part 1/2)

```
/* Fibonacci.c: example demonstrating recursion */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 11/14/04 RD  initial version                 */

#include <stdio.h>

/* function definition */

long fibonacci(long n)
{
   if (n <= 1)  /* base case */
      { return n;
       } /* fi */
   else         /* recursion step */
      { return fibonacci(n-1) + fibonacci(n-2);
       } /* esle */
} /* end of fibonacci */

/* main function */
...
```


Recursion Revisited

- Program example: **`Fibonacci.c`** (part 2/2)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = fibonacci(n);

   /* output section */
   printf("The %ld-th Fibonacci number is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```

Recursion Revisited

- Timed example session: **`Fibonacci.c`**

```
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 41
The 41-th Fibonacci number is 165580141.
2.37 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 42
The 42-th Fibonacci number is 267914296.
3.71 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 43
The 43-th Fibonacci number is 433494437.
5.62 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 44
The 44-th Fibonacci number is 701408733.
8.97 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci
Please enter value n: 45
The 45-th Fibonacci number is 1134903170.
14.26 seconds
%
```


Recursion Revisited

- Example Revisited: Fibonacci series
  - Recursive definition:
    - Base case: $fibonacci(0) = 0,\ fibonacci(1) = 1$
    - Recursion step: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$
  - Mathematical properties:
    - The first two numbers are 0 and 1
    - Every subsequent number is the sum of the previous two
  - ➢ Problem:
    - Program run-time grows exponentially! (factor 1.6, golden ratio)
  - ➢ Idea:
    - If we *remember the previously calculated numbers*, we can calculate the next number immediately!
    - ➢ Whenever a new number is calculated, keep it stored in a **`static`** array in memory
    - ➢ When a number is present in the memory, just look it up

Recursion Revisited

- Program example: **`Fibonacci2.c`** (part 1/3)

```
/* Fibonacci2.c: example demonstrating recursion */
/* author: Rainer Doemer                         */
/* modifications:                                */
/* 11/09/11 RD  version with 'static' memory     */
/* 11/14/04 RD  initial version                  */

#include <stdio.h>

#define MEM_SIZE 100

/* function definition */

...
```


Recursion Revisited

- Program example: **`Fibonacci2.c`** (part 2/3)

```
long fibonacci(long n)
{  static long fib[MEM_SIZE] = {0,1}; /* memory */
   if (n <= 1)        /* base case */
      { return n;
       } /* fi */
   else               /* previouly calculated results */
      { if (n < MEM_SIZE && fib[n])
          { return fib[n];
           } /* fi */
        else          /* recursion step */
          { long f;
            f = fibonacci(n-1) + fibonacci(n-2);
            if (n < MEM_SIZE)
               { fib[n] = f;   /* remember this */
                 } /* fi */
            return f;
          } /* esle */
       } /* esle */
} /* end of fibonacci */
...
```

Recursion Revisited

- Program example: **`Fibonacci2.c`** (part 3/3)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = fibonacci(n);

   /* output section */
   printf("The %ld-th Fibonacci number is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```


Recursion Revisited

- Timed example session: **`Fibonacci2.c`**

```
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 41
The 41-th Fibonacci number is 165580141.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 42
The 42-th Fibonacci number is 267914296.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 43
The 43-th Fibonacci number is 433494437.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 44
The 44-th Fibonacci number is 701408733.
0.00 seconds
% /usr/bin/time -f "%U seconds" ./Fibonacci2
Please enter value n: 45
The 45-th Fibonacci number is 1134903170.
0.00 seconds
%
```