Odds and Ends

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2017/index.html

Topics

1. Generic functions (e.g., swap)
2. Function activation records; static variables
3. Lazy evaluation
4. Measuring time

Global, local variables

- *Global* variables: variables defined outside of the body of any function
  - stored in the data segment
- *Local* variables: variables defined within a function (or block)
  - stored in a region of memory called an *activation record*

Global, local variables

- *Global* variables: variables defined outside of the body of any function
  - stored in the data segment
- *Local* variables: variables defined within a function (or block)
  - stored in a region of memory called an *activation record*

Where are the activation records (AR) stored?

- Simple solution: AR == one fixed block of memory per function
- Better solution: AR allocated / deallocated when function is called / returns
  - variables created when function is called; destroyed when function returns

Global, local variables

- *Global* variables: variables defined outside of the body of any function
  - stored in the data segment
- *Local* variables: variables defined within a function (or block)
  - stored in a region of memory called an *activation record*

Where are the activation records (AR) stored?

- Simple solution: AR == one fixed block of memory per function
- Better solution: AR allocated / deallocated when function is called / returns
  - variables created when function is called; destroyed when function returns
  - need to keep track of *nested* calls
  - function calls behave in *last in first out* manner
    $\Rightarrow$ use *stack* to keep track of ARs

Static variables

Defined within a function, but *not destroyed* when function returns
i.e., retains value across calls to the same function

Static variables

Defined within a function, but *not destroyed* when function returns
i.e., retains value across calls to the same function

Example:

```
void f(void) {
    static int i = 1;
    printf("This function has executed %d time(s)\n",
        i);
    i++;
}
```

Lazy evaluation

Reference: K&R, Section 2.6

- Value of some "compound" Boolean expressions may be determined without evaluating all its sub-expressions.

  Examples:

  - `(x > y) && (a != b)` : if `x` is less than `y`, then the expression is FALSE, irrespective of the value of the second sub-expression
  - `(n > 0) || (i == j)` : if `n` is greater than 0, the expression is TRUE, irrespective of the value of the second sub-expression

- Conditionals / Boolean expressions in C are evaluated from left to right.
- *Evaluation stops as soon as the value of the expression is known. Remaining sub-expressions are not evaluated.*

lazy evaluation

Lazy evaluation

Typical usage:

```
while (i < N && A[i] >= 0) {
       ...
```

- If `i >= N`, `A[i]` is not checked.
- This is useful because checking `A[i] >= 0` when `i >= N` may lead to memory faults.
- In such expressions, `i >= N` serves as a *guard* condition.

Measuring time

- Motivation: to measure the amount of (real) time taken to execute a part of your program
- Relevant headers, types, system calls (cf. `man gettimeofday`)

```
#include <sys/time.h>

struct timeval {
    __time_t tv_sec; /* seconds */
    __suseconds_t tv_usec; /* microseconds. */
};


int gettimeofday(struct timeval *tv, struct
    timezone *tz);
```

stores the number of seconds and microseconds elapsed since the “Epoch” (00:00 of 01.01.1970)

Measuring time

- Auxiliary macro

```
1  #define DURATION(start, end) (end.tv_usec - start.tv_usec))
       + ((end.tv_sec - start.tv_sec) * 1000000
```

- Usage

```
1  struct timeval start_time, end_time;
2
3  /* store the starting time */
4  if (-1 == gettimeofday(&start_time, NULL))
5     ERR_MESG("gettimeofday failed");
6
7  /* code whose running time you want to measure */
8
9  /* store the finishing time */
10 if (-1 == gettimeofday(&end_time, NULL))
11    ERR_MESG("gettimeofday failed");
12
13 /* DURATION computes # microsecs between start and end */
14 printf("%d\n", (int) DURATION(start_time, end_time));
```