Tutorial 8

- malloc / calloc / realloc
- The Data Display Debugger (DDD)
- Exercises on memory management (linked lists & arrays).

malloc, calloc, realloc

- malloc – allocates a chunk of memory

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

e.g.:

```
int *ptr = malloc(10 * sizeof (int));
```

malloc, calloc, realloc

- calloc – allocates and initialize memory (to zero)

```
void *calloc(size_t nelem, size_t elsize);
```

e.g.:

```
int *ptr = calloc(10, sizeof (int));
```

malloc, calloc, realloc

- realloc – grow or shrink a block of memory

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

e.g.:

```
int *new_ptr = realloc(ptr, 250*sizeof(int));
```

Pitfalls

```
int *ptr = malloc(10*sizeof (int));
*ptr = 0;
```

- ptr could be NULL!!! (must check if ptr==NULL)

```
int foo() {
    int *tmp = malloc(100000*sizeof (int));
    ...
}
```

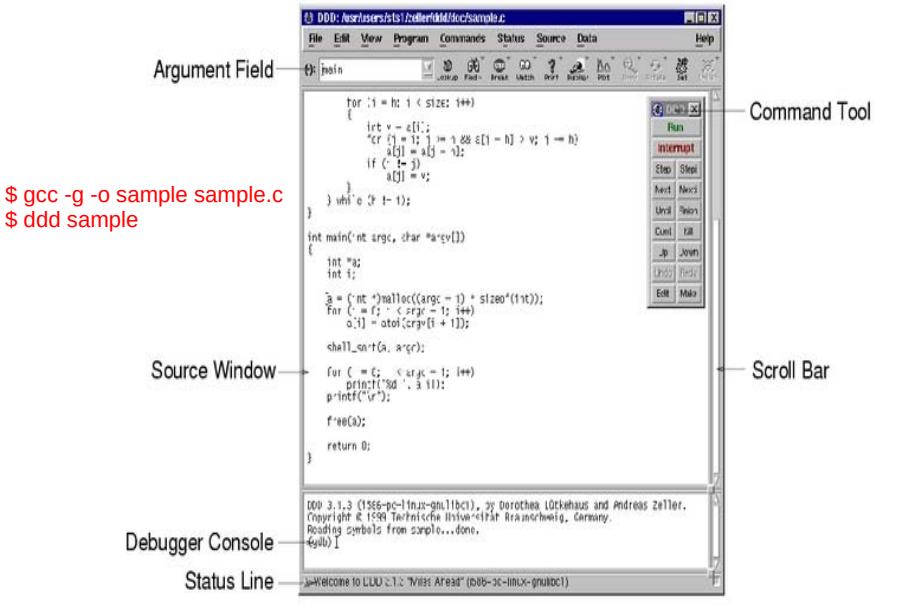
- If tmp is not freed then we have a memory leak!
- Moreover, at each call we allocate a new buffer!!!

Exercise – calloc & realloc

- Compile and run memory.c.
- The program uses calloc and realloc to allocate and resize a buffer.
- Start with a small buffer size and increase it gradually.
- Note that in some cases the address of the array stays the same and in others it changes. Can you explain why?
- What is the memory allocation limit of your system?

DDD – Data Display Debugger

- DDD can do three main kinds of things to help you catch bugs in the act:
  - Start your program with specified parameters and make it stop on specified conditions.
  - Examine what has happened, when your program has stopped.
  - Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.
    - See http://www.gnu.org/manual/ddd/html_mono/ddd.html for a detailed manual

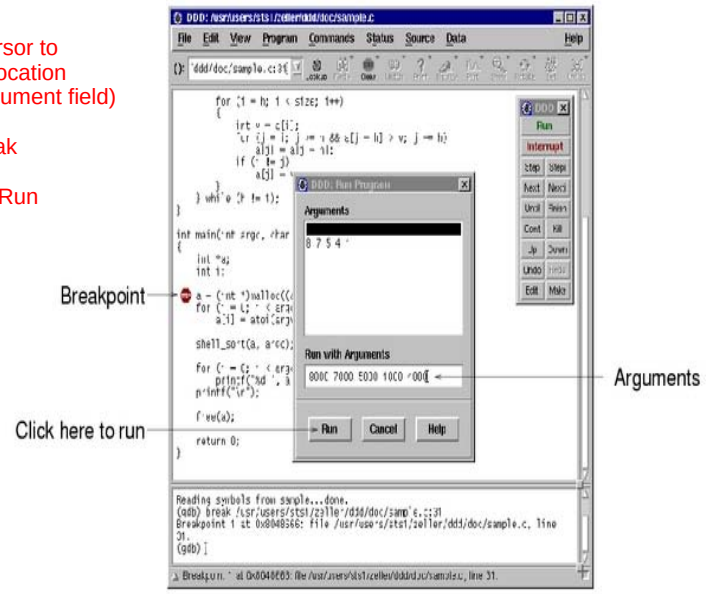


Initial DDD Window

Move the cursor to
the desired location
(note the argument field)

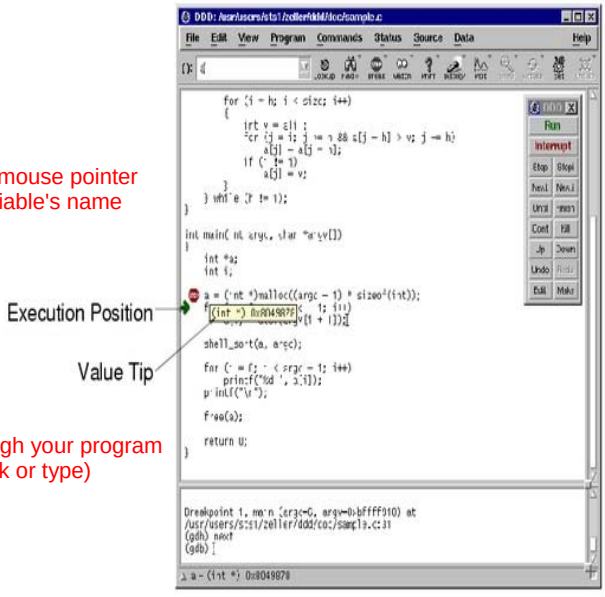
Click on Break

Program => Run



Running the Program

Move the mouse pointer
on the variable's name



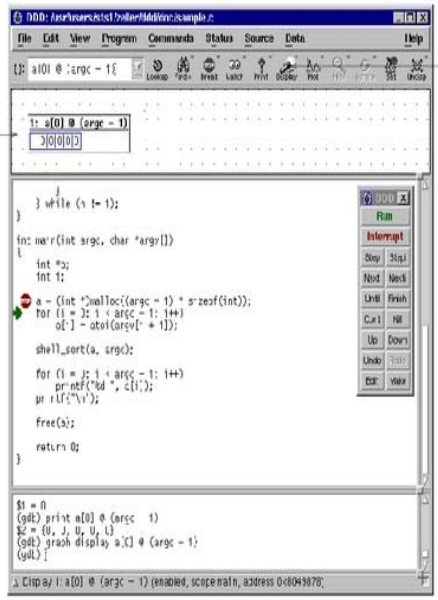
Step through your program
(either click or type)

Viewing Values in DDD

Make sure the variable is in the argument field

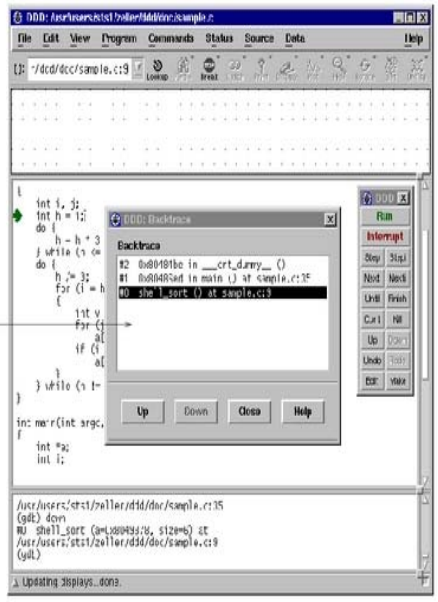
Click on Display

use array[0] @ size for dynamically allocated arrays



Data Window

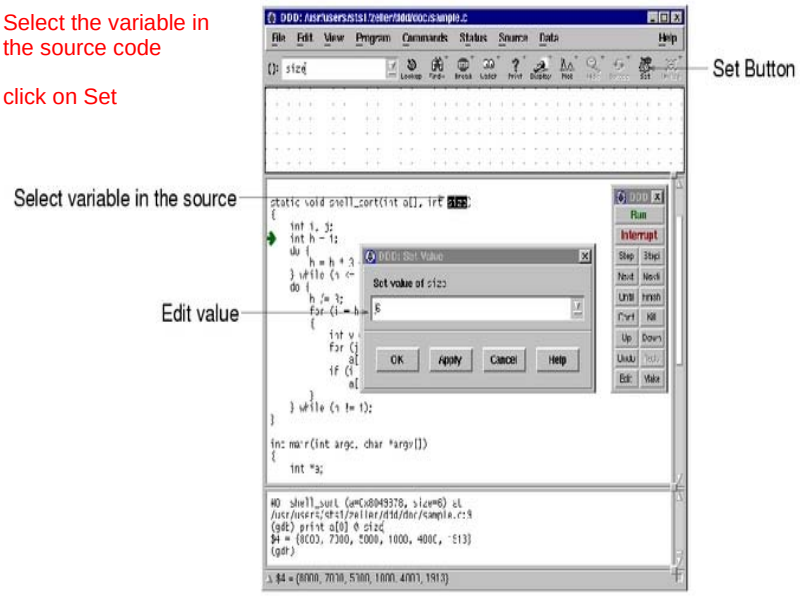
Status => Backtrace



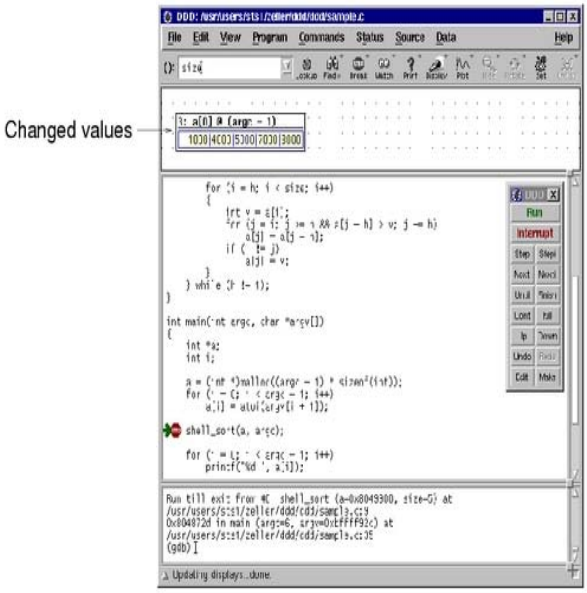
The DDD Backtrace

Select the variable in the source code

click on Set



Setting a Value



Click on Edit to edit your source code with vi

Changed Values after Setting

Exercise – Deallocating linked list

- Say you have a linked list which you are not using anymore. You wrote the following code for freeing up the memory:

```
node *p = head;
while (p!= NULL) {
free(p);
p=p->next;
}
```

This code could crash your program!

(even worse – might work on your system and fail on others, e.g. the TA's...)

Exercise – Deallocating linked list

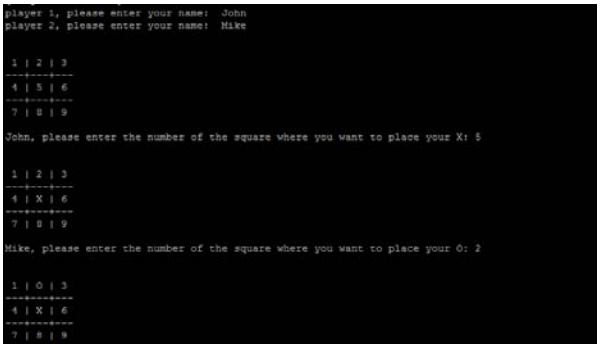
- Therefore, the correct code should use a temporary pointer.

- Go to ex1.c and fix this bug

(ah yes, you'll find another small bug there...)

Exercise – A game of Tic-Tac-Toe

- The code in ex2.c is meant to implement the following game:

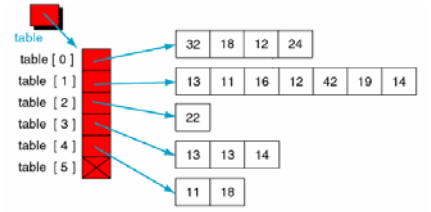


Exercise – A game of Tic-Tac-Toe

- However, some parts were not implemented, and others contain bugs.
- Use the debugger to figure out what's wrong with the code and correct it.
- Hint: focus on pointers and memory allocation.

Exercise – A game of Tic-Tac-Toe

- The following can help you in allocating a 2D array:




```
table = (int **)calloc (rowNum + 1, sizeof(int*));

table[0] = (int*)calloc (4, sizeof(int));
table[1] = (int*)calloc (7, sizeof(int));
table[2] = (int*)calloc (1, sizeof(int));
table[3] = (int*)calloc (3, sizeof(int));
table[4] = (int*)calloc (2, sizeof(int));
table[5] = NULL;
```