CSCI 2132
Software Development

Lecture 28:

Structures and Dynamic Memory Allocation

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- prj-dec2bin and stack example
- Compilation of large programs
- make utility (started)
- make utility
- Using gdb with multi-file programs
- Structures (started)

Structure Example

- Example structure declaration and two variables:

```
struct student {
  int number;
  char name[26];
  char username[11];
} x, y;
```

- Declaring more variables

```
struct student z, *p, first_year[200];
```

- Using dot (`.`) operator:

```
z.number = 123456;
strcpy(z.name, "John King");
```

Example Code

```
x.number = 123;
strcpy(x.name, "Dennis Ritchie");
strcpy(x.username, "dritchie");
y = x;

#define PRINT \
  printf("number:%d\nname:%s\nusername:%s\n\n", \
  x.number, x.name, x.username); \
  printf("number:%d\nname:%s\nusername:%s\n\n", \
  y.number, y.name, y.username);

PRINT
x.number = 456;
strcpy(x.name, "Ken Thompson");
strcpy(x.username, "kthompson");
PRINT
```

Arrow Operator (`->`)

- Arrow operator `p->f` is a short-hand for `(*p).f`
- Example:

```
p = &z;
(*p).number = 222333;
```

- Second line equivalent to:

```
p->number = 222333;
```

- Structures can be used as function parameters
- Not a good practice to pass large structures by value

Dynamic Memory Allocation

- is allocation and release of memory on *heap*
- Very flexible: we control time span and amount of memory
- Using several library functions declared in `stdlib.h`
- Function `malloc` prototype:

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

- Returns a generic pointer to allocated memory block
- or NULL, if no memory available

Malloc Example

- Example:

```
int *p  = (int*) malloc(10000*sizeof(int));
if (p == NULL ) {
   ... /* Error */
}
```

Function `free`

- Used to free memory (deallocate, release)
- Prototype:

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

- Can be used only on blocks allocated by `malloc`
- After deallocation, the pointer becomes a dangling pointer
- No garbage collection; beware of memory leaks