Computer Systems & Low-Level Programming

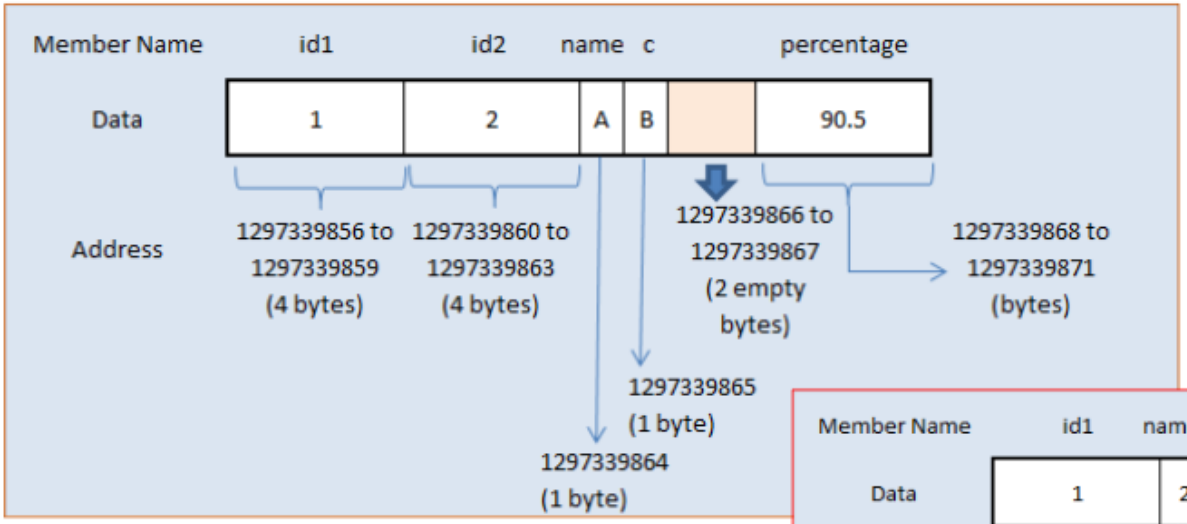
C: structures, files manipulation, memory allocation

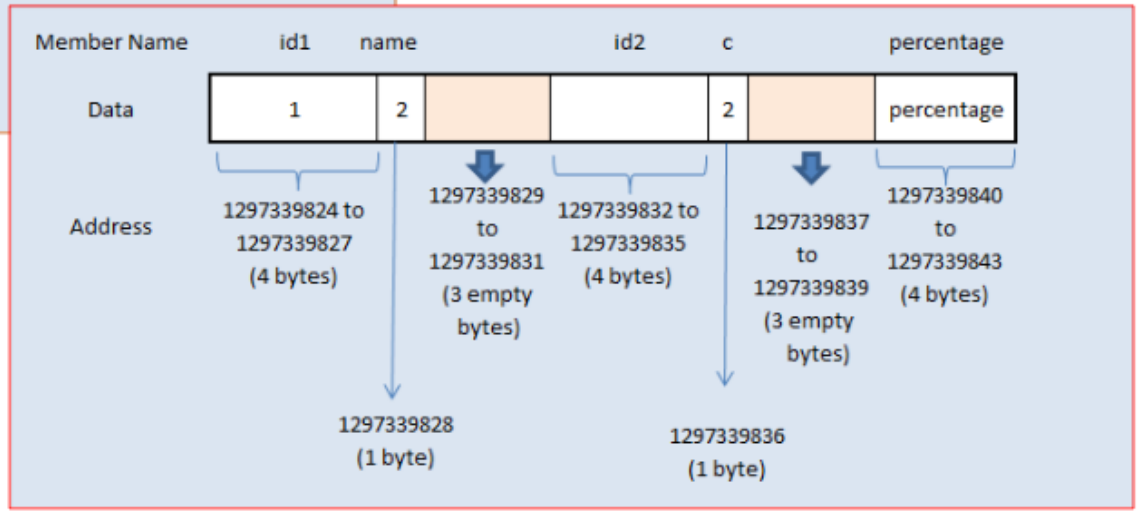
Marija Stanojevic
Spring 2019

Structure

- There are no classes in C, but structures and unions can group variables
- struct course { char name [50]; struct course *requisites d; int id;} - defines structure course which has a pointer to itself (self-referral)
- struct course cProg = {“C and Systems”, NULL, 2107}; - defines cProg variable
- cProg.name = “C Programming and Systems”; -changes name of cProg course
- Structures can used as argument or return type of function
  - struct course rename(struct course cProg);
- Structures can be nested. Arrays of structures are possible:
  - struct schedule {struct course cList[50]; int time[50]}; - creates structure schedule of 50 courses and 50 time points when those courses are happening. Each course is above defined structure
- **Pointer to structure:** struct course *cProgPtr = & cProg;
  - printf(“Course name is %s”, cProgPtr->name); - prints name of course
- **Structure padding:** adding empty space between structure elements to align data in memory. Each element is read from memory in smallest number of 4B

Structure padding example





Typedef, union and enum

- **typedef** struct course cInfo; - defines name **cInfo** for **structure course**
  - cInfo c; - creates variable c of type cInfo, ie. of type struct course
  - cInfo *p; - creates pointer to type cInfo, ie. to struct course.
  - c.id = 2207; p->id = 1076; - changes course id to 2207, then to 1076
- **union** grade {float mean; int max; int min;} - allocates space ONLY for the one (biggest) variable, ie. only one variable can be used at each moment.
  - union grade g; defines variable g of type union grade
  - g.mean = 3.4; sets mean to 3.4; g.max = 5; forgets g.mean and sets max
- **enum dept** {CS = 10, Math = 30, Physics = 50} marks; defines variable **marks** of type **enum dept** with three given values.
  - enum bools {false, true} isCold; - defines boolean behaviour

Files manipulation (1)

- FILE * file; - pointer to a file
- file = fopen(“filename.txt”, “r”); - opens file for reading
- fscanf(file, “%s”, s); - reads string from file and stores it into s
- fclose(file); - closes file after reading (don’t forget)
- To write into file open file for writing: file = fopen(“file.txt”, “w”); fprintf(file, “Hi”);
- To read/write you can also use: fgets/fputs, fgetc/fputc
- If you try to write to non-existing file, it will be created; Otherwise, it is emptied.
- To add data to existing file open it in append mode: file = fopen(“file.txt”, “a”);
- If you try to read from non-existing file or some other error happen when trying to open file fopen() will return NULL.
- To read/write/append binary file use: **“rb”, “wb”, “ab”** mode, respectively

Files manipulation (2)

- fprintf(stderr, “Error in file opening”); - prints error message to screen
- **ferror(file);** - returns non-zero value if error occured on stream file
- **feof(file);** returns non-zero if the end of file is reached
- if (file != NULL) {read/write} - checks if file is actually opened
- while(fgets(file, 100, s) != EOF) {...} - reads lines until end of file is reached
- while(!feof(file)) {...} - does something until you didn’t come till the end of file
- rewind(file); - resets your position to the beginning of file
- fseek(file, 10, SEEK_SET); - sets position in file to 10, SEEK_SET is 0
- ftell(file); - returns the current position in file
- fwrite(ptr, size, len, file); - writes into file from array stored at ptr, size is the size of each element in array and len is number of elements in that array
- fread(ptr, size, len, file); - same as above, but reads from file into ptr array

Memory allocation

- Functions are defined in <stdlib.h>
- `int * ptr = malloc (10 * sizeof(int));` - allocates space for 10 int in memory
- `int * ptr = calloc (10, sizeof(int));` - allocates space for 10 int in memory and sets 0 into each of those 10 positions
- `free(ptr);` - frees memory on which ptr shows, so that it can be used by others
- `realloc(ptr, 5 * sizeof(double));` - gets memory for 5 doubles and frees the rest
- if (ptr != NULL) {...} - only if ptr is not null memory is allocated for you
- Memory allocation happens on Heap
- After `free(ptr);`, you need to do `ptr = NULL;`, otherwise ptr shows on place which is not yours anymore.
- Memory allocated in function is lost once that function finishes.