Parallel Computing with GPUs

Memory
Part 4 - Structures and Binary Files

The University Of Sheffield.

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

❑Structures

  ❑Express a collection of variables as a structure and identify how to access member variables

❑Binary Files

  ❑Apply functions to read and write to binary files

Structures

- A structure is a collection of one or more variables
    - Variables may be of different types
    - Groups variables as a single unit under a single name
- A structure is not the same as a class (at least in C)
    - No functions
    - No private members
    - No inheritance
- Structures are defined using the `struct` keyword
    - Values can be assigned with an initialisation list or through structure member operator ‘.’

```
struct vec{
    int x;
    int y;
};

struct vec v_1 = {123, 456};
struct vec v_2;
v_2.x = 123;
v_2.y = 456;
```

Features of structures

- As with everything, structures are passed by value

```
struct vec make_vec(int x, int y){
    struct vec v = {x, y};
    return v;
}
```

- Pointers to structures use a different member operator
    - ‘->’ accesses member of a pointer to a struct
    - Alternatively dereference and use the standard operator ‘.’

```
struct vec v = {123, 456};
struct vec *p_vec = &v;//CORRECT
p_vec->x = 789;//CORRECT
p_vec.x = 789; //INCORRECT
```

- Declarations and definition can be combined

```
struct vec{
    int x;
    int y;
} v1 = {123, 456};
```

Structure assignment

- ❑Structures can be assigned
  - ❑Arithmetic operators not possible (e.g. `vec_2 += vec_1`)

```
struct vec vec_1 = {12, 34};
struct vec vec_2 = {56, 78};
vec_2 = vec_1;
```

- ❑**BUT** No deep copies of pointer data
  - ❑E.g. if a person `struct` is declared with two `char` pointer members (`forename` and `surname`)

```
struct person paul, imposter;
paul.forename = (char *) malloc(5);
paul.surname = (char *) malloc(9);
strcpy(paul.forename, "Paul");
strcpy(paul.surname, "Richmond");
imposter = paul;     // shallow copy
strcpy(imposter.forename, "John");
printf("Forename=%s, Surname=%s\n", paul.forename, paul.surname);
```

What is the Output?

Structure assignment

- Structures can be assigned
    - Arithmetic operators not possible (e.g. `vec_2 += vec_1`)

```
struct vec vec_1 = {12, 34};
struct vec vec_2 = {56, 78};
vec_2 = vec_1;
```

- **BUT** No deep copies of pointer data
    - E.g. if a person `struct` is declared with two `char` pointer members (`forename` and `surname`)

```
struct person paul, imposter;
paul.forename = (char *) malloc(5);
paul.surname = (char *) malloc(9);
strcpy(paul.forename, "Paul");
strcpy(paul.surname, "Richmond");
imposter = paul;     // shallow copy
strcpy(imposter.forename, "John");
printf("Forename=%s, Surname=%s\n", paul.forename, paul.surname);
```

```
Forename=John, Surname=Richmond
```

Structure allocations

- Structures passed as arguments have member variables values **copied**
  - If member is a pointer then pointer value copied not the thing that points to it (shown on last slide)
  - Passing large structures by value can be quite inefficient

- Structures can be allocated and assigned to a pointer
  - `sizeof` will return the combined size of all structure members
  - Better to pass big structures as pointers

```
struct vec *p_vec;
p_vec = (struct vec *) malloc(sizeof(struct vec));
//...
free(p_vec);
```

Type definitions

- The keyword `typedef` can be used to create ‘alias’ for data types
  - Once defined a `typedef` can be used as a standard type

```
//declarations
typedef long long int int64;
typedef int int32;
typedef short int16;
typedef float vec3f [3];

//definitions
int32 a = 123;
vec3f vector = {1.0f, -1.0f, 0.0f};
```

- `typedef` is useful in simplifying the syntax of `struct` definitions

```
struct vec{
    int x;
    int y;
};
typedef struct vec vec;
vec p1 = {123, 456};
```

Binary File Writing

- size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
    - size_t: size of single object
    - nmemb: number of objects
    - Returns the number of objects written (if not equal to nmemb then error)

```
void write_points(FILE* f, point *points){
  fwrite(points, sizeof(point), sizeof(points) / sizeof(point), f);
}

void main(){
  point points[] = { 1, 2, 3, 4 };
  FILE *f = NULL;
  f = fopen("points.bin", "wb"); //write and binary flags
  write_points(f, points);
  fclose(f);
}
```

Binary file reading

❑size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)

```
void read_points(FILE *f, point *points, unsigned int num_points){
    fread(points, sizeof(point), num_points, f);
}

void main(){
    point points[2];
    FILE *f = NULL;
    f = fopen("points.bin", "rb"); //read and binary flags
    read_points(f, points, 2);
    fclose(f);
}
```

This Lecture (learning objectives)

- ❑Structures
  - ❑Express a collection of variables as a structure and identify how to access member variables

- ❑Binary Files
  - ❑Apply functions to read and write to binary files