Lecture 6
C Programming Language

Summary of lecture 6

- Pointers review
- Casting
- Structures
- Pointers to structures
- Linked List example

Casting

- Casting is a way to force an expression to be evaluated to a certain type
- Example:
  ```
  int j = 6;
  double d = 2.9;
  ```
  - The following three expressions are evaluated to three different values:
  ```
  j/d (== 2.0689)
  (int) (j/d) (==2)
   j/(int)d (==3)
  ```

  - Here we force an argument of a function to be of the correct type:
  ```
  d = sqrt((double)j);
  ```

Casting cont.

- There are cases where we have to declare pointers without prior knowledge about the type they will point to.
- The type void * (pointer to void) is used as a generic pointer type.
  In a mixed type pointer expression, conversion is automatic.
- However, casting is necessary when pointers are accessed.
- Example:

```
int j;
double d, e;
void * pt0 = &j, *pt1 = &d;

e = *pt0 + *pt1;

e = *((int *)pt0) + *((double*)pt1);
```

Structures

- Syntax:
  struct <name>{field_list}
- Struct is a type that is built from several, simpler types
- Struct allowes access to each component
- Almost every operation on built in types (int, float) is legal for structures
  Legal: Arrays of structures, return type of functions
  Illegal: overloading predefined arithmetical or logical operations
- Structures can be nested

Structures cont.

- Example - id cards:

```
struct personal_id {
        int id_number;
        char first_name[15];
        char middle_name[15];
        char last_name[20];
        struct {
             char street[100], city[50];
             unsigned int house_number, zip;
        } address;
        struct personal_id * father;
};
typedef struct personal_id ID ;
ID myself;
myself.first_name = “Aya”;
myself.last_name = “Aner”;
int Check_Relate(ID per1, ID per2) {
  if (strcmp(per1.last_name,per2.last_name)==0)
        return 1;
}
```

Pointer to Struct

- To access data in the struct through a variable use the “.” operator
  myself.first_name = “Aya”;
- You can also define a pointer to a struct, in which case use the “->” operator
- ID per1, per2, *per3;   /* or struct personal_id */
  per1.id_number = 213425;
  per2.id_number = 1113242;
  per3 = (ID*)malloc(sizeof(ID));

  per3->id_number = 2001011;
  or
  (*per3).id_number = 2001011;

- per3->father = &per2;
  strcpy(per3->father->last_name, per1.last_name);
- note: the operators “.” and “->” have the same precedence (the highest), and are associated left to right

More on Struct

- Structures can have pointer elements too:

```
struct personal_id {
        int id_number;
        char * first_name;
        char * middle_name;
        char * last_name;
        struct {
            char *street, *city;
            unsigned int house_number, zip;
        } address;
        struct personal_id * father;
};
```

- The following are equivalent:

```
struct personal_id * myself;
ID * myself;
```

- Same with struct elements:

```
strcpy(myself->last_name,myself->father->last_name);
```

Dynamic Struct Arrays

- Arrays of struct can be fixed or dynamic:

```
ID  myfixedfamily[20];
ID * mydynfamily;
 mydynfamily = (ID*) malloc (sizeof(ID)*20);
```

- The following are equivalent:

```
myfixedfamily[10].first_name = “Aya”;
mydynfamily[10].first_name = “Aya”;
```

Recursive Structures

- Linked List:

```
struct Listitem {
        int number;
        struct Listitem * next;
};
```