

Computer systems engineering I

Nicholas Wickström

IDE, Halmstad University

2009

Outline

C Structures (again)

Länkade datastrukturer

Double linked lists

Bitfält

Functions as arguments in function calls

C Structures (again)

```
struct person {
    char cName[MAX_NAME_LENGTH];
    int nYearOfBirth;
    float fLength; /* in meters */
};

sizeof(struct person) = sizeof(char)*MAX_NAME_LENGTH +
    sizeof(int) +
    sizeof(float);
```

C Structures (again)

```
struct person rFather={"Steve", 1934, 1.88},
              rMother={"Judith", 1937, 1.65};

printf("%s, %d, %2.2f\n", rFather.cName,
       rFather.nYearOfBirth, rFather.fLength);

/* Output: Steve, 1934, 1.88 */
```

```

/* Function for writing a struct person */
void print_person (struct person *prPrintPerson)
{
    printf("%s, %d, %2.2f\n",prPrintPerson->cName,
        prPrintPerson->nYearOfBirth, prPrintPerson->fLength);

/* Note! prPrintPerson->fLength is
    equal to (*prPrintPerson).fLength */
}

/* Called like this: */
    print_person(&rMother);

```

```

struct children {
    struct person rData;
    struct person *prFather;
    struct person *prMother;
} rSon={ "Mike", 1960, 1.92, NULL, NULL};

/* Assign the son's father and mother */
rSon.prFather = &rFather;
rSon.prMother = &rMother;

/* print son's personal data */
print_person(&rSon.rData);
printf("The son, %s, has a father called %s\n",
    rSon.rData.cName,rSon.prFather->cName);

```

Linked data stuctures

```
struct c_list {
    int nInfo;
    struct c_list *rest;
}; /* Simple list object */

struct c_list *prList, *prListObj1, *prListObj2;

struct c_list *alloc_obj(int nData);
void print_list(struct c_list *prTemp);
void put_at_tail(struct c_list **prRefList,
                struct c_list *prObj);
```

```
struct c_list *alloc_obj(int nData)
{
    struct c_list *prListObj;
    prListObj = (struct c_list *) malloc(1 *
        sizeof(struct c_list));

    if(prListObj == NULL) {
        printf("Error: out of memory\n");
        return NULL;
    }
    else {
        prListObj->nInfo = nData; prListObj->rest = NULL;
        return prListObj;
    }
}
```

```
void print_list(struct c_list *prTemp)
{
    struct c_list *prTemp2;
    /* Operate on the temporary pointer not on the list */
    prTemp2 = prTemp;
    while (prTemp2 != NULL) {
        printf("%d\n",prTemp2->nInfo);
        prTemp2 = prTemp2->rest;
    }
}
```

```
void print_list2(struct c_list *prTemp)
{
    /* printing the list can be made recursive */
    /* (not that it is recommended ) */
    if (prTemp != NULL) {
        printf("%d\n", prTemp->nInfo);
        print_list2(prTemp->rest);
    }
}
```

```

void put_at_tail(struct c_list **prRefList,
                struct c_list *prObj) {
    struct c_list *prTemp = *prRefList;
    if (*prRefList != NULL) { /* if list exist */
        while (prTemp->rest != NULL) {
            /* Traverse the list to the end */
            prTemp = prTemp->rest;
        }
        /* We have found the end of the list */
        prTemp->rest = prObj; /* Connect the last node */
    } else {
        /* List empty, ie list _is_ the node provided */
        *prRefList = prObj;
    }
}

```

```

void put_at_tail(struct c_list **prRefList,
                struct c_list *prObj)
{
    while(*prRefList != NULL)
        prRefList = &(*prRefList)->rest;

    prObj->rest = *prRefList;
    *prRefList = prObj;
}

```

```
prList = alloc_obj(0);
prListObj1 = alloc_obj(1);
prListObj2 = alloc_obj(2);

/* Put prListObj1 first in prList1 */
prListObj1->rest = prList;
prList = prListObj1;
```

Double linked lists

```
struct d_list {
    int nInfo;
    struct d_list *prev;
    struct d_list *next;
}; /* Double linked list object */
```

Tree

```
struct t_list {
    int nInfo;
    struct t_list *left;
    struct t_list *right;
}; /* tree list object */
```

Bit fields

```
struct DataOutReg {
    unsigned int PortEnable: 1;
    unsigned int          : 5;
    unsigned int StatusInp : 1;
    unsigned int StatusOut : 1;
    unsigned int Data      : 8;
} rMyReg;

nStatus = rMyReg.StatusInp && rMyReg.StatusOut;
```


Functions as arguments in function calls

```
void myfunc(void)
{
    printf("Here!\n");
}

void doit(void (*func)(void))
{
    printf("In doit: ");
    func();
}
```

```
void main(void)
{
    myfunc();
    doit(&myfunc);
}
```

```
/*
Here!
In doit: Here!
*/
```