CSE2031 Software Tools - Memory Allocation and Structures

Summer 2010

Przemyslaw Pawluk

Department of Computer Science and Engineering
York University
Toronto

May 25, 2010

Notes

What have we done last time?

- Arrays
- Pointers

Notes

What we will do today?

1. Pointers review
2. Dynamic memory allocation
3. Structures

Notes

What do we know about pointers?

Notes

Problem

```
1 int x;
2 scanf("%d", &x);
3 int my_array[x];
```

How can we allocate memory during run time?

The code above is incorrect!

Notes

Solution

Use functions from stdlib.h

Notes

malloc()

```
1 void *malloc(int n);
```

- Allocate memory at run time.
- Returns a pointer to a void if successfully allocated n bytes in the memory
- Returns null if the memory was not allocated.
- The memory block is not initialized.

Notes

calloc()

```
1 void *calloc(int n, int s);
```

- Allocates an array of n elements where each element has size s;
- calloc initializes memory to 0.

Notes

realloc()

```
1 void * realloc(void *ptr, int n);
```

- What if we want our array to grow (or shrink)
- Resizes a previously allocated block of memory.
- ptr must have been returned from either calloc, malloc, or realloc.
- Array may be moved if it could not be extended in its current location.

Notes

free()

```
1 void free(void *ptr)
```

- Releases the memory we previously allocated.
- ptr must have been returned by malloc, alloc, or realloc.

Notes

Dynamic allocation - troubles

```
1 main() {
2 int *x;
3 int size;
4 x=(int*) malloc(size);
5 *x = 20; /* What is wrong? */
6 }
```


Notes

Memory leaks

```
1 int *x;
2 x=(int *) malloc(20);
3 x=(int *) malloc(30);
4 /* What's wrong?*/
```

Memory block allocated in line 2 is lost for ever.
MAY cause problems (exhaust memory).

Notes

Inappropriate use of freed memory

```
1 char *x;
2 x=(char *) malloc(50);
3 free(x);
4 x[0]= A ;
```

Surprise

Surprisingly this code may work on some systems, but in general we cannot predict the result.

Notes

Other issues with freed memory

Freeing unallocated memory

```
1 char *x=NULL;
2 free(x);
```

Freeing "not yours" block

```
1 x=malloc(50);
2 free(x+1);
```

Double freeing

```
1 x=malloc(50);
2 free(x);
3 free(x);
```


Notes

Structures

```
1 struct {
2       float width;
3       float height;
4 } chair, table;
```

```
1 struct dimensions {
2       float width;
3       int height;
4 };
```


Notes

Access and pointers

- Accessing the members is done via '.' operator
- Structs cannot be assigned
- &chair is the address of the variable chair of type struct

Notes

Namespaces

- struct names have their own namespace separate from variables and functions;
- struct member names have their own namespace.

```
1 struct dimesnsion {
2 float width;
3 float height;
4 } height;
5 struct dimension dimension;
```

Notes

Structures and Pointers

Precedence of '.' !!!

```
1 struct simension table, *p;
2 p= &table;
3 *p.width /* INCORRECT */
4 (*p).width; /* CORRECT */
```

You can use '->' operator to access a structure's fields

```
1 p->width;
```

Notes

Initialization of Structures

```
1 struct dimension sofa={2.0, 3.0};
```

Notes

Nested Structures

```
1 struct point {int x, int y;};
2 struct line {
3    struct point a;
4    struct point b;
5 } myline;
```

Notes

Structures and functions

- You can pass structure as arguments of functions
- This is a call-by-value, a copy of the structure is sent to the function

```
1 float get_area(struct dimension d) {
2      return d.width * d.height;
3 }
```

- Structure can be returned from function

```
1 struct dimension make_dim(int width, int height) {
2       struct dimension d;
3       d.width = width;
4       d.height = height;
5       return d;
6 }
```

Notes

Structures and Functions cont.

- It is inefficient to pass large structures to functions, instead use pointers and you can manipulate the same structure.
- Be careful when passing argument using pointer since the pointee is not a copy!

Notes

typedef

We can define a new type and use it later

```
1 typedef struct {
2    int x,y;
3    float z;
4 } newtype;
5 newtype a1,b1,c1,x;
```

Now, newtype can be used just like int, float and any other type in C

Notes

Complex structures – Linked list

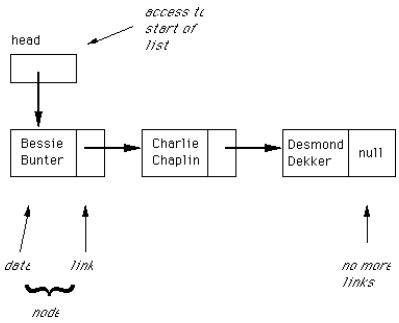
Specification:

- Pointer head points to the first element
- Last element pointer is NULL

```
1 struct list {
2    int data;
3    struct list *link;
4 };
5 struct list *head
```

It is OK to use a pointer to a struct that is declared but not defined

Notes



Notes

Add new node at the end of the list

- Allocate new node.
- Start with head and find the end of the list.
- Assign last link to point to the new node.

Notes

Delete element from the end of the list

- Find the node before the last node.
- Store the address to the last element in the variable.
- Assign NULL to the link field of the element before last.
- Free the element pointed by variable.

Notes

Delete ith element from the list

- Find the i-1 node.
- Store the address to the last element in the variable.
- Assign link of the ith element to the link field of the i-1 element to keep the tail of the list.
- Free the element pointed by variable.

Notes

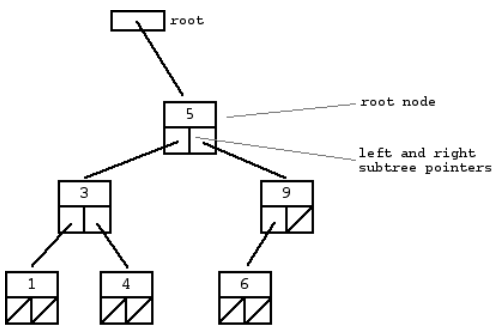
Complex structures – Tree

Specification:

- Each node has up to 2 child nodes
- Each node has 1 parent node
- There is only one element that has no parent node – root node

```
1 struct tree {
2    int data;
3    struct list *lchild;
4    struct list *rchild;
5 };
6 struct list *root
```


Notes



Notes

What we did today

1. Pointers review
2. Dynamic memory allocation
3. Structures

Notes

Next time

- Unions
- Enumerations
- Review of what we did so far.

Notes

Notes