CS 201
Dynamic Data Structures and Function Pointers

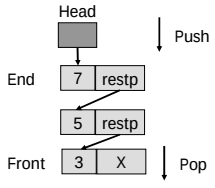
Debzani Deb

Overview

- Queue
- Circular and doubly link list
- Binary Tree
- Function Pointers
- Grading Discussion

Representing a Queue with a linked list

- Like a queue of people waiting
- Push at the Head (i.e at the end of the list).
- Pop from the Bottom (i.e from the front of the list)
- First In First Out (FIFO)



Push on Queue

```
typedef struct list_node_s {
    int digit;
    struct list_node_s *restp;
} list_node_t;
```

Return pointer to the new head of the queue

Pointer to the current head of the queue

- Push is same as stack (at Head)

```
list_node_t * push (list_node_t * qHead, int v) {
    list_node_t * p = (list_node_t *)
    malloc(sizeof(list_node_t));
    p -> digit = v;
    p -> restp = qHead;
    return p;
}
```

Return the new node as the head of the queue

```
int main(void) {
  list_node_t * qHead = NULL;
  /* Function call for push*/
  qHead = push(qHead, 3);
  qHead = push(qHead, 5);
  return 0;
}
```


Pop from Queue (from the bottom)

```
list_node_t * pop (list_node_t * qHead, int * v) {
    list_node_t * qEnd, * qFront = NULL;
    if (qHead -> restp = NULL) {  // Queue has only one element
      *v = qHead ->digit;
       free (qHead);
       return NULL;
    }
    for (qEnd = qHead; qEnd ->restp != NULL; qEnd = qEnd -> restp)
       qFront = qEnd;
    *v = qEnd -> digit;
     qFront -> restp = NULL;
     free(qEnd);
     return qHead;
}
```

Can we write this more efficiently?

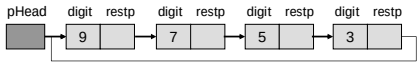
Queue (Summary)

- You could implement it as an array too.
- You could make a hybrid of stack/queue to access at either end.
- Common design for process scheduling, event processing, buffering, input/output etc.
- In our design push is constant time, but pop is O(n) linear time (where n is the number of elements in the queue).
- If we record two pointers (front and end) instead of only one pointer pointing to the head of the list – both push and pop would have constant time.
  - See the implementation in your textbook.

Circular and double linked list

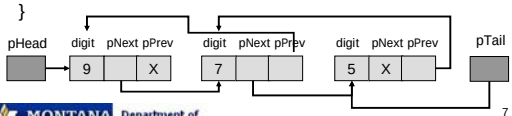
- Circular linked list

```
pHead -> restp -> restp -> restp -> restp = pHead;
```



- Double linked list

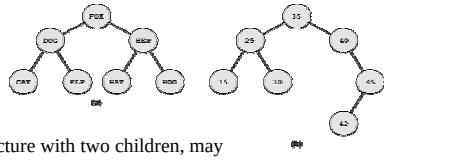
```
struct dblLink {
   int digit;
   struct dblLink * pNext, pPrev;
}
```



Trees

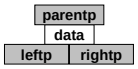
- Non linear data structures
  - Elements can have two or more children.
  - Useful for sorting, graph algorithms, searching etc.
- A simple example – binary tree
  - Each element has 0-2 children.
    - Elements with no children are called a leaf.
  - Each element has 0-1 parents.
    - Elements with no parents are called a root.

Binary Tree



- Structure with two children, may maintain pointer to the parent node too.

```
typedef struct node_s {
   int data;
   struct node_s *leftp, *rightp;
   // struct node_s *parentp;
} node_t;
```



Binary Search Tree

- Useful for sorting
  - Can make an invariant like: all elements in leftp are less than data, all elements in rightp are greater than data.
- Useful for searching
  - With an invariant such as above, you can find an element in the tree in $O(\log_2 n)$ time.
  - Binary search tree is a very common abstraction to maintain.
  - Many algorithms are there.

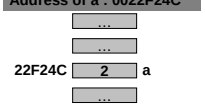
Function Pointers

Addressing Functions

- Variables reside in program's memory space.
- Address of a variable can be found with & operator.

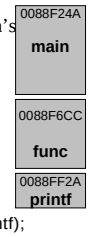
```
int main(void) {
   int a = 2;
   printf("Adddress of a:
          %p\n", &a);
   return 0;
}
```

Address of a : 0022F24C



- Function resides in program's memory space as well.

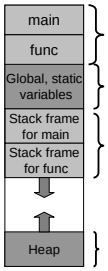
```
void func(int b) {
   printf("b = %d\n",b);
}
int main(void) {
   printf("Adddress of \n
          main():  %p\n
          func():  %p\n
          printf():  %p\n",
          &main, &func, &printf);
   return 0;
}
```



Address of
main(): 0088F24A
func(): 0088F6CC
Printf(): 0088FF2A

& operator returns function address

Functions in address space



Code — Machine language instructions of a function are found here.

Static data

Stack Frames — Variables declared inside a function can be found here.

Dynamically allocated variables

Order of the memory components (code, static data, stack and heap) may be different depending on the memory models utilized by the compiler.

Function Pointers (1)

- By analogy with ordinary variables, the address of a function can be stored in a pointer variable : **the function pointer.**

Pointer to Variable	**Pointer to Function**
declaration: int i;	declaration: int f (int arg);
pointer	pointer
declaration: int *ip = &i;	declaration: int (*fp) (int arg) = &f;

parentheses are essential for maintaining precedence

Argument name is optional

- Defines a function pointer fp, pointing to a function with one integer argument and returning an integer.
- Function pointer and the function it points to must be compatible in return type and argument list.

Function Pointers (2)

```
void f (int);
void (*fp) (int);
```

fp may points to a function of the form void f (int).

- **Like Arrays, Function name is an address of the beginning of the function.** Therefore:

```
fp = f;
fp = &f;
```

Both assign the same location, i.e. the beginning address of the code of function f to the function pointer fp.

- Creating your own function pointer type can be useful sometime.

```
typedef int (*fp_t) (int);
fp_t fptr = &func;
```

The identifies fp_t is now a synonym for the type "pointer to a function that takes an int argument and returns int".

Using Function Pointers

```
double ari (int a, int b) { return (a+b)/2; } // Arithmetic mean
double geo (int a, int b) { return sqrt (a*b); } // Geometric mean
double (*fp) (int a, int b) = NULL;
```

- Assigning an address to a function pointer:

```
fp = &ari;      /* assignment using addresses */
fp = geo;       /* short form */
```

- Calling a function using a function pointer:

```
double result = (*fp)(10, 20);   /* explicit dereferencing */
double result = fp(10, 20);      /* short form, still same */
```

- Array of function pointers:

```
double (*fp[2]) (int, int) = {ari, geo};
for(i=0; i<2; i++)
    printf("Mean %d: %.2f\n", i, fp[i](10, 20);
```

Mean 0: 15

Mean 1: 14.14

Passing a function pointer as an argument to another function

- Function pointers can be used to pass functions as arguments to another function.

```
double mean (int a, int b, double (*fp) (int, int) ) {
   double result = (*fp) (a,b);
   return result;
}
```

- Function is called by passing just a function name (i.e. a pointer to the address of the function code in short form)

```
double ari_mean = mean (10, 20, ari);
double geo_mean = mean (10, 20, geo);
```

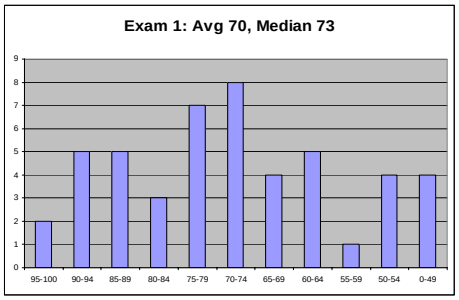
- http://www.eskimo.com/~scs/cclass/int/sx10a.html

Grading

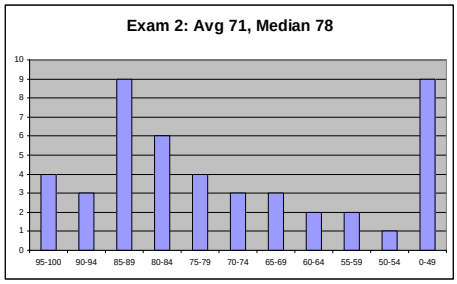
Grading

- How Final grading will be done?
  - Labs : 50%
  - Quiz: 2% each : Total 4 (Last one is coming…)
  - Exam 1 & 2 : Each 11% (Look for the scaled grades)
  - Final Exam: 20%
- A, B, C & D will be defined by curve fitting.

Exam 1: Histogram (on scaled data)



Exam 2: Histogram (on scaled data)



Email Suggestions (summarized)

- Person 1: Writing code that actually compile is too much considering the time constraints, instead suggested pseudo code/algorithm.
- Person 2: Coding is difficult and error prone, suggested critical thinking questions, extra credit lab.
- Person 3: Suggested extra credit based on gdb.
- Person 4 & 5: Everybody gets same time for both tests, no extra credit, scale/curve the result.
- Person 6 : Median, scale/curve the result.

Important !!!

- Special Exam: There will be an exam early next week.
  - Syllabus same as exam 2, Standard same as exam 2.
  - You have the option of replacing your Exam 1 & 2 grade with the result of that single test.
  - Can appear only if you have the probability of getting a F or D.
  - Email me with the intent within today.
- Your TA (Fuad) will be in Bozeman from this Friday.
  - Make appointment with him by sending email and clear all the pending labs by next week.
- Check lab 7 & 9 grades next Monday.