Discussion 5

Mid-term

- Style
  - True/False, Multiple choices, Fill in the blank
    - Some questions from quizzes
  - Short answer
- Material Covered: Based on everything covered through yesterday's class
- You will have up to 2 hours.

C: Data Types

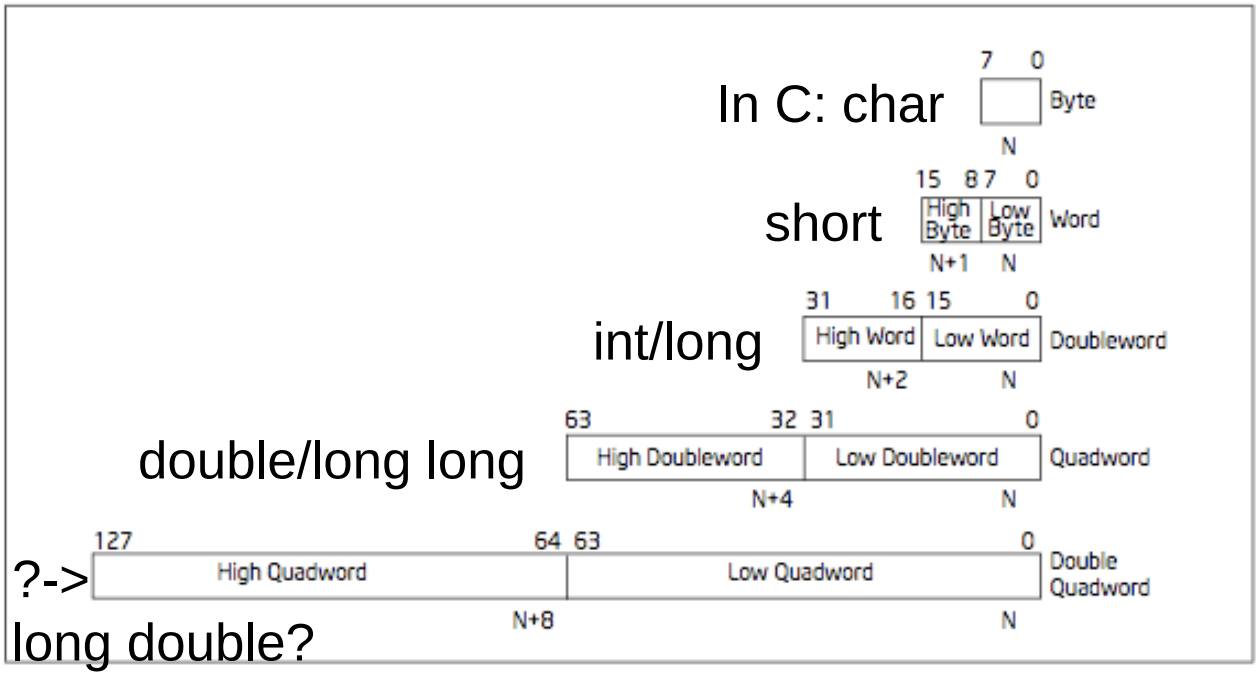


Figure 4-1. Fundamental Data Types

i-clicker question

- In a 32-bit CPU machine, what is the range of a signed int?
  A. -2^07 to +2^07-1
  B. -2^15 to +2^15-1
  C. -2^31 to +2^31-1
  D. -2^63 to +2^63-1

i-clicker question

- What is the range of a signed 3-bit number?
  A. -2^1 to +2^1-1
  B. -2^2 to +2^2-1
  C. -2^3 to +2^3-1
  D. -2^4 to +2^4-1

Common bit task

Get ith bit

- Left shift 1 over i bits
- Perform AND
- Compare result to 0

Set ith bit

- Left shift 1 over i bits
- Perform OR

Example

- You are given two 32-bit numbers, N and M, and two bit positions, i and j. Write a method to insert M into N such that M starts at bit j and ends at bit i. You can assume that the bits j through i have enough space to fit all of M. That is, if M=10011, you can assume that there are at least 5 bits between j and i. You would not, for example, have j=3 and i=2, because M could not fully fit between bit 3 and bit 2.

  Example

  Input: N = 10000000000, M = 10011, i = 2, j = 6

  Output: N = 10001001100

Array

- Memory layout of multidimensional arrays
- Array initialization
  - float banana [5] = {…}
  - float honeydew[] = {…}
  - short cantaloupe[2][5] = {
    {…},
    {…},
    };
  - int rhubarb[][3]={{…}, {…}, };

Array

- Only two things can be done to array:
  - determine its size
  - obtain a pointer to element 0 of the array.
- All other operations are done with pointers

  T[] a

  T* p = a;

  *(p+i) = a[i]

  addr(ptr+i) = addr(ptr) + [sizeof(T)*i]

Group Activity: String

strStr() Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack. Complete the following C code with the blanks filled in.

```
int strStr(char *haystack, char *needle) {
    int hLen = strlen(haystack);
    int nLen = strlen(needle);
    if(hLen == 0)
        return -1;
    if(nLen == 0)
        return 0;
    for(int i=0; i<hLen-nLen; i++) {
        for(int j=0; ; j++) {
            if(_____)
                return i;
            if(*(haystack+i+j) !=_____)
                break;
        }
    }
    return -1;
}
```

Dynamic data structure: Linked List

- Define the structure of linked list node

```
struct node
{
   int data;
   struct node* next;
};
```

- Create a node:

```
struct node* new_node = (struct node*) malloc(sizeof(struct node));
```

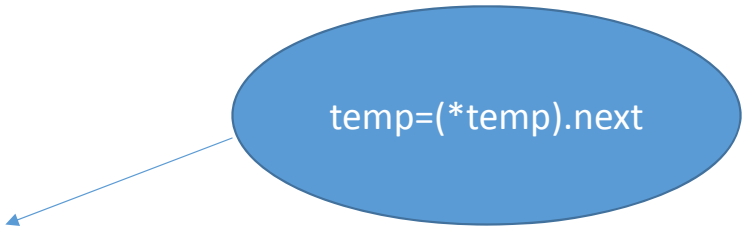
- Free a node

```
free(new_node);
```

Dynamic data structure: Linked List

- Traverse a linked list:

```
struct node* temp=head;
while(temp!=NULL)
{
    // do something
    temp= temp->next;
}
```



Dynamic data structure: Binary Search Tree

- Define a binary search tree node

```
struct node
{
  int key_value;
  struct node *left;
  struct node *right;
};
```

- Free all nodes in a binary search tree

```
void destroy_tree(struct node *leaf)
{
  if( leaf != 0 )
  {
      destroy_tree(leaf->left);
      destroy_tree(leaf->right);
      free( leaf );
  }
}
```

Dynamic data structure: Binary Search Tree

- Create a node

```
struct node* leaf = (struct node*) malloc( sizeof( struct node ) );
```

- In-order traversal

```
void inorder(struct node *temp) {
  if (temp != NULL) {
    inorder(temp->lchild);
    // do something
    inorder(temp->rchild);
  }
}
```

i-clicker question

- Where is the C dynamic memory allocation functions defined?

A. stdio.h
B. string.h
C. stdlib.h
D. ctype.h

i-clicker question

- What is the best way to create a linked list node using malloc?
  A. struct node* new_node = (struct node*) malloc(sizeof(struct node));
  B. struct node* new_node = malloc(sizeof(struct node));
  C. struct node new_node = malloc(sizeof(struct node));
  D. struct node* new_node = (struct node*) malloc(10000);

i-clicker question

root is the root node of a binary search tree.

What does ”foo” do?

```
struct node* foo(struct node* root, int data)
{
   if (root == NULL) {
        struct node* node = (struct node*)
       malloc(sizeof(struct node));
       node->data  = data;
         return node;
else
   {
      if (data <= root->data)
         root->left  = foo(root->left, data);
      else
         root->right = foo(root->right, data);
      return root;
   }
}
```

A. Insert a new node with the value data
B. Delete a node with the value data
C. Search a node with the value data
D. Destroy the tree represented by root