Unit 1

- was developed by Dennis M. Ritchie in early 1970's
- unlike python, C does not have an interactive mode.

comments: /* */ or //

data type: unlike python. once associated a data type with specific variable, the type cannot be changed.

int char (one character)
double/float void (no data type attached)

Basic Structure:

importing a library, we use the '#include<>' statement

#include<stdlib.h> — standard C library
#include<stdio.h> — standard input/output library

```
main( ) {
    int x;
   double y;
   char one_character;
   program_statement_1;   Every statement ends in ;
   program_statement_2;
   f1(1, 2, 3);  —— a sample function call }

f1(int x, int y, int z)
{
   int w;
   program statement;
```

fz( ); — a call to other function
}

few notes on printf( )

printf("... formatting string...", variables, to, print, separated, by, comas);

Formatting options:

%d —— Prints a decimal (integer) number
%f —— Prints a float point number
%c —— Prints a single character
%s —— Prints a string
%g —— Prints a floating point number using scientific notation
%p —— Prints a pointer (this will be formated as a hexadecimal value)

\n —— New line
\t —— Print a 'tab'
\% —— Print a '%' symbol
\\ —— Print a '\' symbol

e.g. printf("My variables are: %d, %f, %s\n", a, b, c);

My variables are: 10, 3.1415, Hello world

ex.

```
int i,j;  // We can decla
          // line, separa

for (i=0; i<10 ; i=i+1)
{
   for (j=0; j<i; j=j+1)
   {
    printf("%d, ",j);
   }
   printf("\n");
}
```

output:

0,
0, 1,
0, 1, 2,
⋮
0, 1, 2, ⋯, 8,

eg int x=3.14
print("%d", x);
output: 1903008824

|| logical or

&& logical and

! logical not

Unit 2

- Declaring a variable does not initialize it or set is to 0
- locker are assigned based on where there is a box of right size that is empty (might contains junk) (different data types need different amount of space)

function declaration:

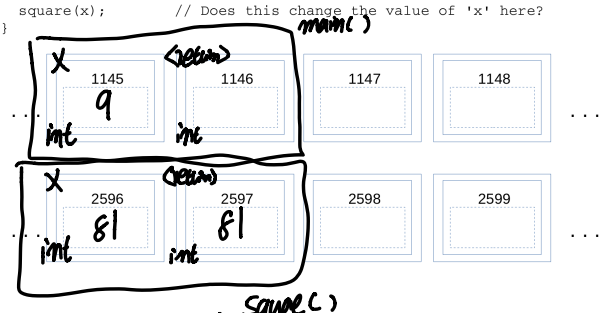
```
int square(int x)
{
}
```

when function call done its job, memory releases.

- each function has its own space and cannot be directly accessed by any other code.
- passing argument makes a copy.

Ex

```
#include<stdio.h>

int  square(int x)
{
  x=x*x;
  return x;
}
int main()
{
  int x;
  x=9;
  square(x);        // Does this change the value of 'x' here?
}
```



local variable vs global variables

```
int x= 6;
float y= 9.1;
printf("%f", x)
printf("%d", y)
```

result ???? ???

This is Not type cast

```
int x=3;
float y;
y=x;
printf("%f", y)
```

result 3.000000 (6 0s)

This is type cast

```
float x=9.999;
int y;
y=x;
printf("%d", y)
```

result 9

This is a type cast (dropping the fractional part)

```
int my_array[10] // an array of 10 integer values
```

where my_array[0] is the first entry

my_array[9] is the last entry

reserves consecutive box and the size is fixed and same type

print a str without strcpy

```
char    a_string[1024];

a_string[0]='H';
a_string[1]='e';
a_string[2]='l';
a_string[3]='l';
a_string[4]='o';
a_string[5]='\0';

printf("%s\n",a_string);
```

Will produce:

```
Hello
```

print a str with strcpy

```
#include<stdio.h>
#include<string.h>      // - This is the C library for string management

int main()
{
    char   a_string[1024];

char   a_string[1024];

// Let's initialize a string - we can do that with the strcpy() function
strcpy(a_string, "This is a message for those learning C\n");

// Let's now concatenate another string to what we already have
strcat(a_string, "Don't forget to practice using strings!\n");

// Let's print out what we have stored

    printf("%s\n",a_string);
}
```

```
...\a.exe
This is a message for those learning C
Don't forget to practice using strings!
```

Strcpy

Strcat

Strlen()

Strcmp()

Pointer

```
#include<stdio.h>

int main()
{
  int my_int;
  int *p=NULL;      // A pointer to an int!

  my_int=10;
  printf("My int is: %d\n",my_int);

  my_int=15;
  printf("My int is: %d\n",my_int);

  p=&my_int;
  *(p)=21;
  printf("My int is: %d\n",my_int);
}
```

If you compile and run the above, you get:

```
\a.out
My int is: 10
My int is: 15
My int is: 21
```

pointer to arrays

```
#include<stdio.h>

void square_array(int array[10])
{
     int j;

     for (j=0; j<10; j=j+1)
     {
       array[j]=array[j]*array[j];
     }
}


int main()
{
    int my_array[10];
    int i;

    for (i=0; i<10; i=i+1)
    {
         my_array[i]=i;
    }

    square_array(my_array);

    for (i=0; i<10; i=i+1)
    {
        printf("%d squared equals %d\n",i,my_array[i]);
    }
}
```

Passing a pointer

pointer arithmetic (since memory box is consecutive)

```
#include<stdio.h>

int main()
{
     int  array[10];
     int  *p=NULL;

     p=&array[0];      // Get address of the first entry in the array

     *(p+0)=5;         // Set the contents of array[0] to 5
     *(p+4)=10;        // Set the contents of array[4] to 5

     printf("array[0] is %d, array[4] is %d\n",array[0],array[4]);
}
```

in array, the name of the array is equivalent to a pointer to the first entry in the array

You should always make sure that:

- Pointers are initialized to *NULL* when you declare them - this will save you no end of trouble as you can check whether the pointer has actually been assigned to something or not.
- Every function that receives a pointer as a parameter ***must check*** that the input pointer is not *NULL*. You can not access a *NULL* pointer and trying to do so will crash your program.
- When using pointers and offsets to access data, you must ensure the offsets used are within bounds for the array you're indexing into.

Unit 3

```
#define MAX_STRING_LENGTH 1024
```

two compound data type cannot be compared

Getting user input

- Numeric types (ints and floats)

```
int x, y;
float pi;
scanf("%d %d %f", &x, &y, &pi);
getchar();
```

usually come together

- strings

```
char stringy[1024];
fgets(stringy, 1024, stdin);
```

name — max number — standard input

in the case, fgets read 1023 chars since the last should be '\0'

List Abstract Data Type (List ADT)

- *Declaring* a new (empty) list
- *Adding* items to an existing list
- *Removing* items from a list
- *Searching* for a specific data item

since it does not specify how the operations are implemented.

Creating node on-demand is called dynamic memory allocation

```
/*
   CSC A48 - Unit 3 - Containers, ADTs, and Linked Lists

   This program implements a linked list of restaurant reviews.
   The program allows the user to enter as many reviews as needed,
   to print the existing reviews, and when finished, it releases
   all memory allocated to the list before exiting.

   (c) 2018 - F. Estrada & M. Ahmadzadeh.
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX_STRING_LENGTH 1024

typedef struct Restaurant_Score
{
       char restaurant_name[MAX_STRING_LENGTH];
       char restaurant_address[MAX_STRING_LENGTH];
       int score;

} Review;

typedef struct Review_List_Node
{
     Review  rev;
     struct Review_List_Node *next;
} Review_Node;

Review_Node *new_Review_Node(void)
{
     Review_Node *new_review=NULL;       // Pointer to the new node

     new_review=(Review_Node *)calloc(1, sizeof(Review_Node));
     // Initialize the new node's content with values that show
     // it has not been filled. In our case, we set the score to -1,
     // and both the address and restaurant name to empty strings ""
     // Very importantly! Set the 'next' pointer to NULL

     new_review->rev.score=-1;
     strcpy(new_review->rev.restaurant_name,"");
     strcpy(new_review->rev.restaurant_address,"");
     new_review->next=NULL;
     return new_review;
}
```

Standard input output .h
Standard library .h
Type
Type
no input and returns a pointer.
type cast
type
initialize (wipes-out junk)
calloc ( )
1. find available place
2. reserve space
3. wipes-out junk
4. return pointer

```
Review_Node *insert_at_head(Review_Node *head, Review_Node *new_node)
{
     // This function adds a new node at the head of the list.
     // Input parameters:
     //        head : The pointer to the current head of the list
     //        new_node:  The pointer to the new node
     // Returns:
     //         The new head pointer

     new_node->next=head;
     return new_node;
}

void print_reviews(Review_Node *head)
{
      Review_Node   *p=NULL;  // Traversal pointer

      p=head;                 // Initialize the traversal pointer to
                              // point to the head node
      while (p!=NULL)
      {
        // Print out the review at this node
         printf("*********************************************\n");
         printf("Restaurant Name: %s\n",p->rev.restaurant_name);
         printf("Restaurant Address: %s\n",p->rev.restaurant_address);
         printf("Restaurant Score: %d\n",p->rev.score);
         printf("*********************************************\n");
         // Update the traversal pointer to point to the next node
         p=p->next;
      }
}

void delete_list(Review_Node *head)
{
      Review_Node *p=NULL;
      Review_Node *q=NULL;

      p=head;
      while (p!=NULL)
      {
           q=p->next;
           free(p);
           p=q;
      }
}
```

```
Review_Node *delete_by_name(Review_Node *head, const char name_key[])
{
      // This function removes the node from the link list that contains the
      // review with a matching restaurant name.

      Review_Node *tr=NULL;
      Review_Node *pre=NULL;

      if (head==NULL) return NULL;     // Empty linked list!

      // Set up the predecessor and traversal pointers to point to the first
      // two nodes in the list.
      pre=head;
      tr=head->next;

      // Check if we have to remove the head node
      if (strcmp(head->rev.restaurant_name, name_key)==0)
      {
           free(pre);       // Delete the first node in the list
           return tr;       // Return pointer to the second node (new head!)
      }

      while(tr!=NULL)
      {
          if (strcmp(tr->rev.restaurant_name, name_key)==0)
          {
               // Found the node we want to delete
               pre->next=tr->next;          // Update predecessor pointer
               free(tr);                    // remove node
          }
          tr=tr->next;
          pre=pre->next;
      }
Review_Node *search_by_name(Review_Node *head,\
                            const char name_key[MAX_STRING_LENGTH])
{
      // Look through the linked list to find a node that contains a
      // review for a restaurant whose name matches the 'name_key'
      // If found, return a pointer to the node with the review. Else
      // return NULL.

      Review_Node   *p=NULL;  // Traversal pointer

      p=head;
      while (p!=NULL)
      {
        if (strcmp(p->rev.restaurant_name,name_key)==0)
        {
            // Found the key!
             return p;
        }
         p=p->next;
      }
      return NULL;      // The search key was not found!
}
```

```
int main()
{
      Review_Node  *head=NULL;
      Review_Node  *one_review=NULL;
      char name[MAX_STRING_LENGTH];
      char address[MAX_STRING_LENGTH];
      int score;
      int choice=1;

      while (choice!=4)
      {
            printf("Please choose one of the following:\n");
            printf("1 - Add a new review\n");
            printf("2 - Print existing reviews\n");
            printf("3 - Update review for one restaurant\n");
            printf("4 - Exit this program\n");

            scanf("%d",&choice);
            getchar();

            if (choice==1)
            {
               // Get a new review node
               one_review=new_Review_Node();

               // Read information from the terminal to fill-in this review
               printf("Please enter the restaurant's name\n");
               fgets(name, MAX_STRING_LENGTH, stdin);
               printf("Please enter the restaurant's address\n");
               fgets(address, MAX_STRING_LENGTH, stdin);
               printf("Please enter the restaurant's score\n");
               scanf("%d",&score);
               getchar();

               // Fill-in the data in the new review node
               strcpy(one_review->rev.restaurant_name,name);
               strcpy(one_review->rev.restaurant_address,address);
               one_review->rev.score=score;

               // Insert the new review into the linked list
               head=insert_at_head(head,one_review);
            }
            else if (choice==2)
            {
               print_reviews(head);
            }
            else if (choice==3)
            {
               printf("Which restaurant's score do you want to update?\n");
               fgets(name,MAX_STRING_LENGTH,stdin);
               one_review=search_by_name(head,name);
               if (one_review==NULL)
               {
                  printf("Sorry, that restaurant doesn't seem to be in the
list\n");
               }
               else
              {
                 printf("Please enter the new score for the restaurant\n");
                 scanf("%d",&one_review->rev.score);
                 getchar();
              }
           }
      }

      // User chose #3 - Release memory and exit the program.
      delete_list(head);
      return 0;
}
```

Bitwise operation

- only integrals
- & and $\frac{11}{10}{}_{10}$
- | or $\frac{10}{01}{}_{11}$
- ^ exclusive or $\frac{10}{01}{}_{11}$ $\frac{11}{00}$
- << shift
- >>
- ~ $\sim\frac{10010}{01101}$

Unit 4

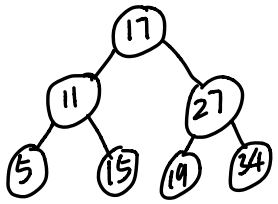
Bubble sort: $O(N^2)$

Quick sort: average $O(N \log N)$ worse $O(N^2)$

```
Bst_Node *BST_insert( root, new_review)
{
 if (root == NULL) return new_review;
 if (strcmp(new_review->rev.name, root->rev.name) < 0)
 {
   root->left = insert(root->left, new_review);
 }
else
{
  root->right=insert(root->right, new_review);
}
return root;
```

```
BST_Node * BST_search (BST_Node *root, name[1024])
{
  if (root == NULL)  return NULL;
  if (strcmp(root->rev.name, name) == 0)
  {
    return root;
  }
  if ( strcmp(name, root->rev.name) < 0)
  {
     return  BST_search (root->left, name);
  }
  else
  {
     return BST_search (root->right, name);
  }

BST_Node * find_successor (BST_Node * right_child)
{
  BST_Node* P = NULL;
  P = right_children;
  while ( P->left  != NULL)
  {
    P = P->left
  }
  return  p;
```

```
BST_Node *BST_delete(BST_Node*root, char name[1024])
{
  BST_Node* tmp = NULL;
  if (root == NULL) return NULL;
  if (Strcmp(root->rev.name, name) == 0)
  {
    if (root->left == NULL && root->right == NULL)
    {
      free(root);
      return NULL;
    else if (root->right == NULL)
    {
      tmp = root->left;
      free(root);
      return tmp;
    }
    else if (root->left == NULL)
    {
      tmp = root->right;
      free(root);
      return tmp;
    }
    else
    {
      tmp = find_successor(root->right);
      root->rev.name = tmp->rev.name
      root->right = BST_delete(root->right, tmp->rev.name)
      return root;
    }
    if (strcmp(name, root->rev.name) < 0)
      root->left = BST_delet(root->left, name)
    else
      root->right = BST_delete(root->right, name)
    return root;
}
```



in-order 5 11 15 17 19 27 34 from left to right →
pre-order 17 11 5 15 27 19 34 ↓ created a copy
post-order 5 15 11 19 34 27 17 ↑ delete a BST

Unit 5

	Adjacency list	Adjacency matrix
PROS	Space efficient when large V small E	No need to travetsal, easy access
CONS	traversal list might be slow	Not space effient
check 2 node connect	O(V)	O(1)
insert node	O(1)	$O(V^2)$
remove node	O(E)	$O(V^2)$
insert edge	O(1)	O(1)
remove edge	O(V)	O(1)

Sort on array

① Split 1st one every time heap sort
N split and N merge results $O(N^2)$

② random quick sort
O(N log N) : log N split N merge or $O(N^2)$

③ half merge sort
O(N log N):

DFS and BFS

Unit 6

good property

- Modularity (individual module; easy to debug; easy to understand)
- Reusability (high chances to be reused)
- Extendibility (easy to extend)
- Maintainability (be organized)
- correctness (thoroughly tested)
- Efficiency (good algorithm)
- Openness (when possible, contribute)
- Privacy and security

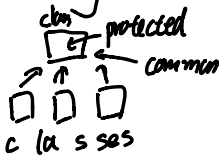
Object Oriented Programming

data in class : member variables
functions in class: class methods

```
typedef struct ListNodeStruct
{
     char string[1024];
     struct LastNodeStrucy *next;
} ListNode;

class StringList
{
     // Member variables
     private:
          ListNode *head;
          int list_length;

     // Class methods
     public:
                StringList()
                {
                  head=NULL;
                  list_length==0;
                }
                ~StringList();
          void insert_string(char string[1024]);
          void delete_string(char string[1024]);
          ListNode search(char string[1024]);
          void clear_list();
          int  get_length();
     private:
          void set_length(int l);

};
```

Method Overloading (multiple version)

Inheritance



polymorphism

```
BST_Node *insert(root, new_node)
{
  if (root==NULL) return new_node;
  if (strcmp(new_node->rev.name, root->rev.name)<=0
      root->left=insert(root->left,new_node);
  else
      root->right=insert(root->right,new_node);
  return root;
}

BST_Node* search(root, name[20])
{
  if (root == NULL) return NULL;
  if(strcmp(root->rev.name, name)==0)
      return root;
  if(strcmp(name, root->rev.name)<0)
      return search(root->left, name);
else
      return search(root->right,name);
}
BST_Node *find_successor( right_children)
{
 BST_Node *P=NULL;
 P= right_children;
while (p->left != NULL)
    P=P->left;
return P;
}
```

```
BST_Node *delete(root, name[1024])
{
  BST_Node *tmp = NULL;
  if(root == NULL) return NULL;
  if(strcmp(root->rev.name, name)==0)
      if (root->left==NULL && root->right==NULL)
          free(root);
          return NULL;
     else if (root->left == NULL)
          tmp= root->right;
          free(root);
          return tmp;
     else if (root->right==NULL)
          tmp=root->left;
          free(root);
          return tmp;
     else
          tmp=find_successor(root->right);
          strcpy(root->rev.name, tmp->rev.name);
                    ⋮
          root->right= delete(root->right, tmp->rev.name)
          return root
  if(strcmp(name, root->rev.name)<0)
      root->left=delete(root->left, name);
  else
      root->right=delete(root->right, name);

 return root;


void  in_order(root)


in_order(root->left);
printf.....
in_order(root->right);
```