Starting with C

- C Language Tutorial
  - [http://phy.ntnu.edu.tw/~cchen/ctutor.pdf](http://phy.ntnu.edu.tw/~cchen/ctutor.pdf)

- C Compilation

```c
#include <stdio.h>
#include <math.h>
void main() {
    int angle_degree;
    double angle_radian, pi, value;
    printf ("\nCompute a table of the sine function\n\n");
    pi = 4.0*atan(1.0);
    printf ( " Value of PI = %f \n\n", pi );
    printf ( " angle Sine \n" );
    angle_degree=0; /* initial angle value */
    while ( angle_degree <= 360 ) {
        angle_radian = pi * angle_degree/180.0 ;
        value = sin(angle_radian);
        printf ( " %3d %f \n ", angle_degree, value);
        angle_degree = angle_degree + 10;
    }
}
```
Pointers and variables

float x;
float x, y;
float* px;
float* px1;

x = 6.5;
px = &x;
y = *px;

float x, y;
float* px, *fp, *fp2; /* fp and fp2 are ptrs to float */
x = 6.5; /* x now contains the value 6.5 */
printf("Value of x is %f, address of x %ld \n", x, &x);
fp = &x; /* fp now points to location of x */
printf("Value in memory location fp is %f\n", *fp);
*fp = 9.2;
printf("New value of x is %f = %f \n", *fp, x);
*fp = *fp + 1.5;
printf("Final value of x is %f = %f \n", *fp, x);
y = *fp;
fp2 = fp;
printf("Transfered value into y = %f and fp2 = %f \n", y, *fp2);
Self-learning:

- Arrays
- Functions (passing / returning pointers)
- Command line arguments
- i/o from/to files

Structure
- named structure and typedef (next two slides)
- nested structures
- Array of structures
- Structures and pointers (pointer arithmetic)

Dynamic allocation
- Linked list of structures

```c
main() {
    struct {
        char initial; /* last name initial */
        int age; /* child’s age */
        int grade; /* child’s grade in school */
    } boy, girl;

    boy.initial = 'R';
    boy.age = 15;
    boy.grade = 75;
    girl.age = boy.age - 1;
    girl.grade = 82;
    girl.initial = 'H';

    printf("%c is %d years old and got a grade of %d\n", girl.initial, girl.age, girl.grade);
    printf("%c is %d years old and got a grade of %d\n", boy.initial, boy.age, boy.grade);
}
```
```c
struct child {
    char initial; /* last name initial */
    int age;    /* child’s age */
    int grade;  /* child’s grade in school */
};

struct child boy, girl;

typedef struct {
    char initial; /* last name initial */
    int age;     /* child’s age */
    int grade;   /* child’s grade in school */
} kid;

kid boy, girl;

... 
boy.initial = 'R';
girl.grade = 82;

printf("%c is %d years old and got a grade of %d\n",
girl.initial, girl.age, girl.grade);
```

More

- An array of “kid”
- Typed: indexing and pointer arithmetic
- What about a single liked list of “kid”
main()
{
    struct {
        char initial;
        int age;
        int grade;
    } kids[12];

    int index;
    for (index = 0; index < 12; index++) {
        kids[index].initial = 'A' + index;
        kids[index].age = 16;
        kids[index].grade = 84;
    }
    kids[3].age = kids[5].age = 17;
    kids[2].grade = kids[6].grade = 92;
    kids[4].grade = 57;
    for (index = 0; index < 12; index++)
        printf("%c is %d years old and got a grade of %d\n",
               kids[index].initial, kids[index].age, kids[index].grade);
}

main()
{
    struct {
        char initial;
        int age;
        int grade;
    } kids[12], *point;

    int index;
    for (index = 0; index < 12; index++) {
        point = kids + index;
        kids[index].initial = 'A' + index;
        kids[index].age = 16;
        kids[index].grade = 84;
    }
    ...
    for (index = 0; index < 12; index++) {
        point = kids + index;
        printf("%c is %d years old and got a grade of %d\n",
//               kids[index].initial, kids[index].age, kids[index].grade);
(*point).initial, kids[index].age, point->grade);
    }
}
main( ) {
    struct animal {
        char name[25]; /* The animals name */
        char breed[25]; /* The type of animal */
        int age; /* The animals age */
        struct animal *next; /* a pointer to another record of this type */
    } *point, *start, *prior; /* this defines 3 pointers, no variables */
    int index;

    /* the first record is always a special case */
    start = (struct animal *)malloc(sizeof(struct animal));
    strcpy(start->name,"general");
    strcpy(start->breed,"Mixed Breed");
    start->next = NULL;
    prior = start;

    /* a loop can be used to fill in the rest once it is started */
    for (index = 0;index < RECORDS;index++) {
        point = (struct animal *)malloc(sizeof(struct animal));
        strcpy(point->name,"Frank");
        strcpy(point->breed,"Laborador Retriever");
        point->age = 3;
        point->next = NULL; /* point this "next" to NULL */
        prior->next = point /* point last "next" to this record */
        prior = point; /* this is now the prior record */
    }
    ...

    /* A Linked List example */

    ... now print out the data described above */
    point = start;
    do {
        prior = point->next;
        printf("%s is a %s, and is %d years old.\n", point->name,
        point->breed, point->age);
        point = point->next;
    } while (prior != NULL);

    /* good programming practice dictates that we free up the */
    /* dynamically allocated space before we quit */
    point = start; /* first block of group */
    do {
        prior = point->next; /* next block of data */
        free(point); /* free present block */
        point = prior; /* point to next */
    } while (prior != NULL); /* quit when next is NULL */
}
Dynamic Allocation

- dynamically allocated variables are simply variables that do not exist when the program is loaded, but are created dynamically as they are needed.

- It is possible, using these techniques, to create as many variables as needed, use them, and deallocate their space for use by other variables.

```c
main() {
    struct animal {
        char name[25];
        char breed[25];
        int age;
    } *pet1, *pet2;
    pet1 = (struct animal *)malloc(sizeof(struct animal));
    strcpy(pet1->name,"General");
    strcpy(pet1->breed,"Mixed Breed");
    pet1->age = 1;
}

pet2 = pet1; /* pet 2 now points to the above data structure */
pet1 = (struct animal *)malloc(sizeof(struct animal));
strcpy(pet1->name,"Frank");
strcpy(pet1->breed,"Labrador Retriever");
pet1->age = 3;

/* now print out the data described above */
printf("%s is a %s, and is %d years old." ,pet1->name,pet1->breed,
pet1->age);
printf("%s is a %s, and is %d years old." ,pet2->name,pet2->breed,
pet2->age);
free(pet1); /* this frees up one structure */
free(pet2); /* this frees up one more structure */
}
```