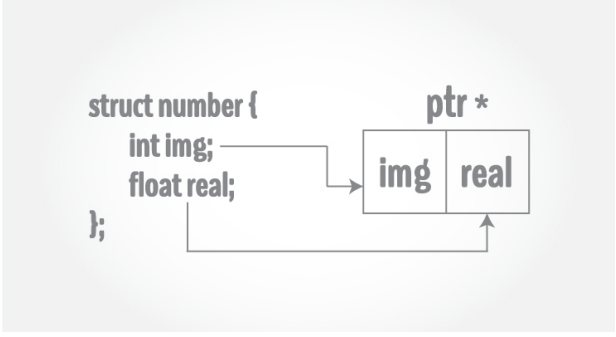
CS1100 – Introduction to Programming

Instructor:

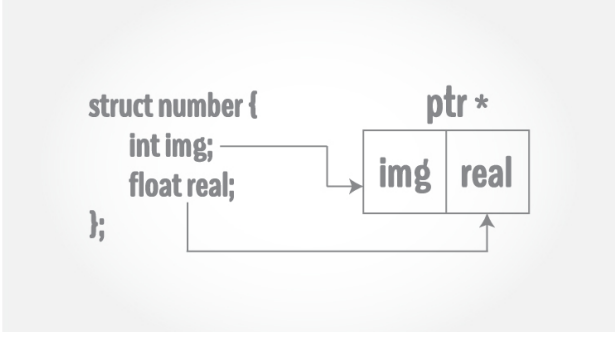
Shweta Agrawal (shweta.a@cse.iitm.ac.in)

Lecture 30

Pointers to Structures

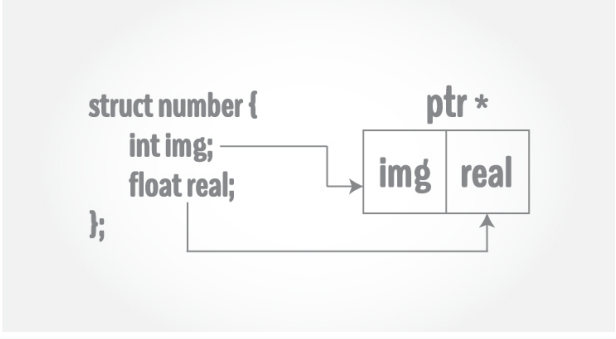


Pointers to Structures




```
#include "stdio.h"
struct number {
    int img;
    float real;
};
int main()
{
    struct number *ptr;
    printf("%d %f\n", ptr->img, ptr->real);
}
```

Pointers to Structures

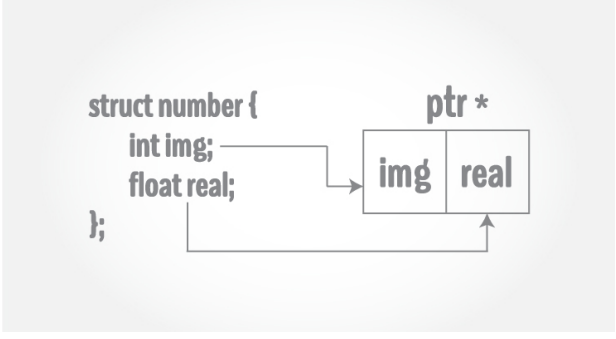



```
#include "stdio.h"
struct number {
    int img;
    float real;
};
int main()
{
    struct number *ptr;
    printf("%d %f\n", ptr->img, ptr->real);
}
```

Accessing an element of the structure pointed to by ptr :

- via pointer dereferncng : `(*ptr).img` and `(*ptr).real`

Pointers to Structures

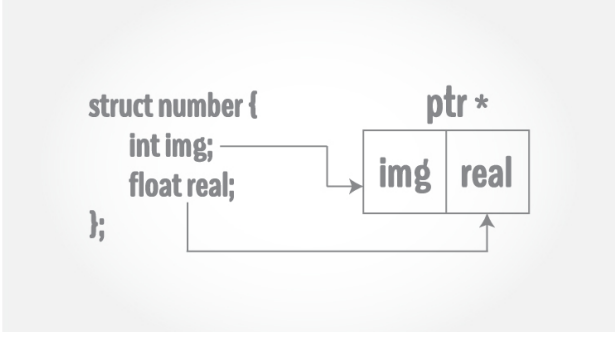



```
#include "stdio.h"
struct number {
    int img;
    float real;
};
int main()
{
    struct number *ptr;
    printf("%d %f\n", ptr->img, ptr->real);
}
```

Accessing an element of the structure pointed to by ptr :

- via pointer dereferncng : `(*ptr).img` and `(*ptr).real`
- Neater method : use `ptr->img` and `ptr->real` respectively. The operator ->is minus sign followed by greater than symbol.

Pointers to Structures




```
#include "stdio.h"
struct number {
    int img;
    float real;
};
int main()
{
    struct number *ptr;
    printf("%d %f\n", ptr->img, ptr->real);
}
```

Accessing an element of the structure pointed to by ptr :

- via pointer dereferncng : `(*ptr).img` and `(*ptr).real`
- Neater method : use `ptr->img` and `ptr->real` respectively. The operator ->is minus sign followed by greater than symbol.
- This will cause segmentation fault. Why?

Pointers to Structures : Accessing the members

```
#include <stdio.h>
struct number {
    int img;
    float real;
};

int main()
{
    struct number *ptr;

    struct number num;
    num.img = 10;
    num.real = 0.56;

    ptr = &num;
    printf("Via Num : %d %f\n",num.img,num.real);
    printf("Via *ptr. : %d %f\n",(*ptr).img,(*ptr).real);
    printf("Via ptr-> : %d %f\n",ptr->img,ptr->real);
}
```

Pointers to Structures: Allocation

```
#include "stdio.h"
#include "stdlib.h"
struct number {
    int img;
    float real;
};
int main()
{
    struct number *ptr=NULL;
    ptr = (struct number *)
      malloc (1*sizeof(struct number));
    ptr->img = 5;
    ptr->real = 5.0;
    printf("%d %f\n", ptr->img, ptr->real);
}
```

Precedence and Association

- Both . and -> associate left to right.
- They are at top of precedence hierarchy
- Example : If we have :
  `struct rectangle r, *rp = r;`
  The following forms are equivalent:

```
r.pt1.x                (r.pt1).x
rp->pt1.x              (rp->pt1).x
(*rp).pt1.x
```

`typedef` in C

- Do not like `float` being used for fractions data type?

`typedef` in C

- Do not like `float` being used for fractions data type?
  You are not alone.

`typedef` in C

- Do not like `float` being used for fractions data type?
  You are not alone. Good news : There is a fix !.

`typedef` in C

- Do not like `float` being used for fractions data type?
  You are not alone. Good news : There is a fix !.
- You can do `typedef` to rename float to your favorite keyword.

`typedef` in C

- Do not like `float` being used for fractions data type?
  You are not alone. Good news : There is a fix !.
- You can do `typedef` to rename float to your favorite keyword.
- Syntax : `typedef float fraction;`

`typedef` in C

- Do not like `float` being used for fractions data type?
  You are not alone. Good news : There is a fix !.
- You can do `typedef` to rename float to your favorite keyword.
- Syntax : `typedef float fraction;`
- Then if you use `fraction x;` it is same as writing `float x;`.

`typedef` in C

- Do not like `float` being used for fractions data type?
  You are not alone. Good news : There is a fix !.
- You can do `typedef` to rename float to your favorite keyword.
- Syntax : `typedef float fraction;`
- Then if you use `fraction x;` it is same as writing `float x;`.
- This has more implications:You can do typedef for structures!.

typedef in C

- Do not like `float` being used for fractions data type? You are not alone. Good news : There is a fix !.
- You can do `typedef` to rename float to your favorite keyword.
- Syntax : `typedef float fraction;`
- Then if you use `fraction x;` it is same as writing `float x;`.
- This has more implications:You can do typedef for structures!.

```
struct student {
    char rollNumber[6];
    char name[20];
    int age;
    int program;
};
typedef struct student STUDENT;
```

typedef in C

- Do not like `float` being used for fractions data type? You are not alone. Good news : There is a fix !.
- You can do `typedef` to rename float to your favorite keyword.
- Syntax : `typedef float fraction;`
- Then if you use `fraction x;` it is same as writing `float x;`.
- This has more implications:You can do typedef for structures!.

```
struct student {
    char rollNumber[6];
    char name[20];
    int age;
    int program;
};
typedef struct student STUDENT;
```

```
int main() {
    STUDENT S1,S2;
    S1 = {"CS15B1","Mahendar",18,1);
    S2 = S1;
}
```

Precedence & Associativity of operators

Precedence order	Operator	Associativity
1	( ) [ ] →	Left to right
2	++ -- - (unary) ! ~ * & sizeof	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >>	Left to right
6	< <= > >=	Left to right
7	== !=	Left to right

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	increments len not p; same as `++(p->len)`
`(++p)->len`	

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	increments len not p; same as `++(p->len)`
`(++p)->len`	increments p before accessing len
`p++->len`	

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	increments len not p; same as `++(p->len)`
`(++p)->len`	increments `p` before accessing len
`p++->len`	increments `p` after accessing len
`*p->str`	

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	increments len not p; same as `++(p->len)`
`(++p)->len`	increments `p` before accessing len
`p++->len`	increments `p` after accessing len
`*p->str`	fetches whatever `str` points to
`*p->str++`	

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	increments len not p; same as `++(p->len)`
`(++p)->len`	increments `p` before accessing len
`p++->len`	increments `p` after accessing len
`*p->str`	fetches whatever `str` points to
`*p->str++`	increments `str` after accessing.
`(*p->str)++`	

Practicing Associativity of -> and .

Given the declaration `struct {int len;char *str;} *p;`

Expression	Action
`++p->len`	increments len not p; same as `++(p->len)`
`(++p)->len`	increments `p` before accessing len
`p++->len`	increments `p` after accessing len
`*p->str`	fetches whatever `str` points to
`*p->str++`	increments `str` after accessing.
`(*p->str)++`	increments whatever `str` points to.

Code Examples - 1

```
#include <stdio.h>
typedef struct complex {
    float real;
    float imag;
} complex;

complex add(complex n1, complex n2);

int main() {
    complex n1, n2, result;

    printf("For 1st complex number \n");
    printf("Enter the real and imaginary parts: ");
    scanf("%f %f", &n1.real, &n1.imag);
    printf("\nFor 2nd complex number \n");
    printf("Enter the real and imaginary parts: ");
    scanf("%f %f", &n2.real, &n2.imag);

    result = add(n1, n2);

    printf("Sum = %.1f + %.1fi", result.real, result.imag);
    return 0;
}

complex add(complex n1, complex n2) {
    complex temp;
    temp.real = n1.real + n2.real;
    temp.imag = n1.imag + n2.imag;
    return (temp);
}
```

Code Examples - 2

```
#include <stdio.h>
struct TIME {
   int seconds;
   int minutes;
   int hours;
};
void differenceBetweenTimePeriod(struct TIME t1, struct TIME t2, struct TIME *diff);
int main() {
   struct TIME startTime, stopTime, diff;
   printf("Enter the start time. \n");
   printf("Enter hours, minutes and seconds: ");
   scanf("%d %d %d", &startTime.hours, &startTime.minutes, &startTime.seconds);
   printf("Enter the stop time. \n");
   printf("Enter hours, minutes and seconds: ");
   scanf("%d %d %d", &stopTime.hours, &stopTime.minutes, &stopTime.seconds);
   differenceBetweenTimePeriod(startTime, stopTime, &diff);
   printf("\nTime Difference: %d:%d:%d - ", startTime.hours,  startTime.minutes, startTime.seconds);
   printf("%d:%d:%d ", stopTime.hours, stopTime.minutes, stopTime.seconds);
   printf("= %d:%d:%d\n", diff.hours, diff.minutes, diff.seconds);
   return 0;
}
void differenceBetweenTimePeriod(struct TIME start, struct TIME stop,  struct TIME *diff) {
   while (stop.seconds > start.seconds) {
      --start.minutes;
      start.seconds += 60; }
   diff->seconds = start.seconds - stop.seconds;
   while (stop.minutes > start.minutes) {
      --start.hours;
      start.minutes += 60;  }
   diff->minutes = start.minutes - stop.minutes;
   diff->hours = start.hours - stop.hours;
   }
```

Code Examples - 3

```
#include <stdio.h>
#include <stdlib.h>
struct course {
  int marks;
  char subject[30];
};

int main() {
  struct course *ptr;
  int noOfRecords;
  printf("Enter the number of records: ");
  scanf("%d", &noOfRecords);

  // Memory allocation for noOfRecords structures
  ptr = (struct course *)malloc(noOfRecords * sizeof(struct course));
  for (int i = 0; i < noOfRecords; ++i) {
    printf("Enter subject and marks:\n");
    scanf("%s %d", (ptr + i)->subject, &(ptr + i)->marks);
  }

  printf("Displaying Information:\n");
  for (int i = 0; i < noOfRecords; ++i) {
    printf("%s\t%d\n", (ptr + i)->subject, (ptr + i)->marks);
  }

  free(ptr);

  return 0;
}
```

Code Examples - 4

```
#include <stdio.h>
#include <stdlib.h>
struct person {
   int age;
   float weight;
   char name[30];
};

int main()
{
   struct person *ptr;
   int i, n;
   printf("Enter the number of persons: ");
   scanf("%d", &n);
   ptr = (struct person*) malloc(n * sizeof(struct person));

   for(i = 0; i < n; ++i) {
       printf("Enter first name and age respectively: ");
       scanf("%s %d", (ptr+i)->name, &(ptr+i)->age);  }

   printf("Displaying Information:\n");
   for(i = 0; i < n; ++i)
       printf("Name: %s\tAge: %d\n", (ptr+i)->name, (ptr+i)->age);

   return 0;
}
```