Lecture 9: Structures and Pointers

Recall: Each variable has a type (`int, float, double,..`)

Example: What does the following program print?

```
#include <stdio.h>
void main()
{
  int i=4,j=3;
  float x=4,y=3;

  printf("%d\n", i/j);
  printf("%f\n", x/y);
}
```

Structure: Collection of variables (possibly of different types) grouped under a single name

(Provide a way for users to define "customized" types)

Example: a structure for storing student information

```
#include <stdio.h>
struct student
    {
       char name[30];
       int section;
    };
void main()
{
   int i;
   struct student person1;
   person1.section = 1;
   printf("%d", person1.section);
}
```

```
struct student
       { char name[30];
           int section;
       };
struct student x;
```

defines a variable x with the following structure

Arrays of Structures

```
#include <stdio.h>
#define N 150
struct student
    { char name[30];
       int section;
       int current_grade;
    };
void main()
{
   int i;
   struct student x={"a student", 1, 10};
   struct student 126takers[N];

   for(i=0; i <N; i++)
    {
     126takers[i]= x;
    }

}
```

Example2: A structure for representing vectors

```
#include <stdio.h>
struct vector
       {
         float x;
         float y;
       };
struct vector add(struct vector u,
                  struct vector v)
{
 struct vector w;
 w.x = u.x + v.x;
 w.y = u.y + v.y;
 return w;
}

void main()
{
  struct vector a = {5.1, 11.0}, b;

  b = add(a, a);
  printf("%f, %f", b.x, b.y);
}
```

Example 2: Dealing with Fractions

(numbers like $\frac{p}{q}$)

```
#include <stdio.h>
struct fraction
       {
        int num;
        int denom;
       };
struct fraction multiply(struct fraction a,
                         struct fraction b)
{
  fraction y;
  y.num = a.num * b.num;
  y.denom = a.denom* b.denom;
  return y;
}
void main()
{
  fraction x, y, z;
  x.num =5; x.denom =7;
  y.num = 3; y.denom=8;
  z = multiply(x, y);
  printf("%d\n", z.num);
}
```

Note: A structure can be used to define new structures

```
struct  student
            { char name[30];
              int section;
            };
struct studentpair
            {
             struct student student1, student2;
            };
```

What does the following declaration do?

```
struct funny
            {
              int field1;
              struct funny field2;
            };
```

Question: How can functions "share" variables?

COS126/128: Spring'96

What does the following program print?

```
#include <stdio.h>
void swap(int x, int y)
{
  int temp;
  temp = x;
  x = y;
  y = temp;
}
void main()
{
 int a=5, b=10;
 swap(a, b);
 printf("a =%d, b=%d", a, b);
}
```

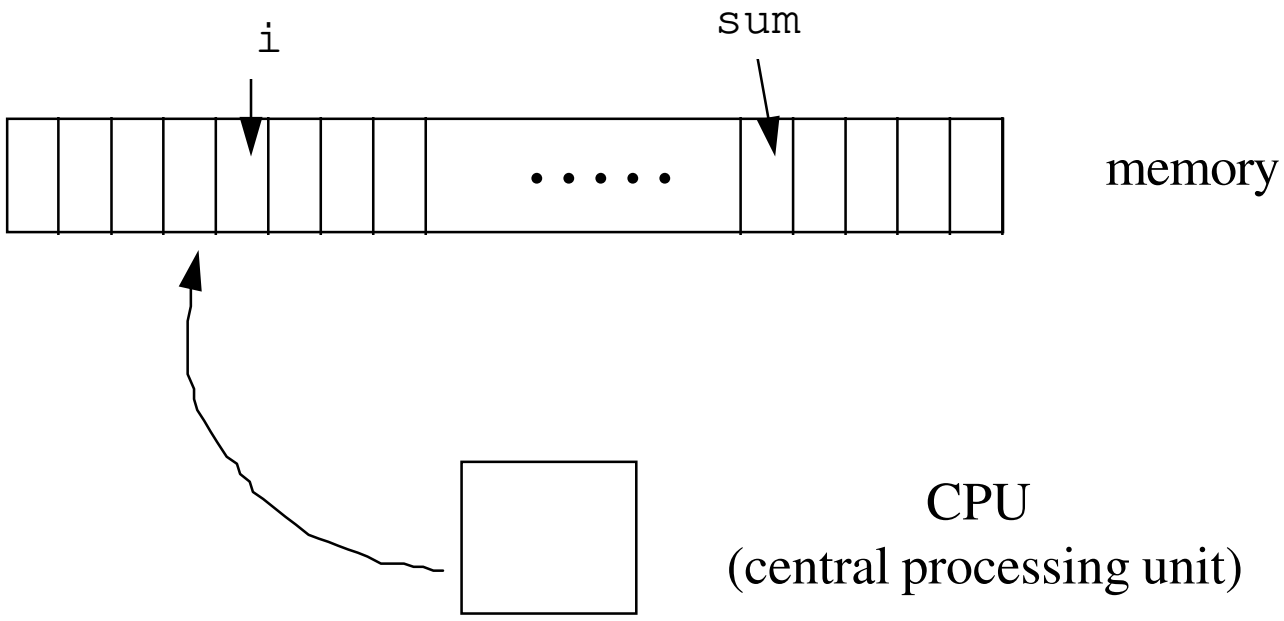
Correct Implementation of Swap( )

```
#include <stdio.h>
void swap(int *x, int *y)
{
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
void main()
{
 int a=5, b=10;
 swap(&a, &b)
 printf("a =%d, b=%d", a, b);
}
```

If `a` is a variable, `&a` is the address of `a`

How variables are stored inside the computer

(a rough picture)




```
int sum=2, i=5;
sum = sum + i;
....
```

"address" of a variable = the location at which the variable's contents are stored.

In C: `&i` denotes address of variable `i`

Pointer Variable : A variable whose only allowable value is an address

How to declare and use pointer variables:

```
int *y;     /* y is a pointer to an integer*/
int i=5;      /*  i is an integer */
y = &i;
*y = 7;
printf("%d", i);
```

Summary:

`&i` = address of variable i

`*y` = value stored in the location pointed to by y

Pointers on a machine with 2^8 words of memory (each address = 8 bits = 2 Hex letters)

```
int i= 5;
int *y;
y = &i;
*y = 7;
printf("%d", i);
printf("%d", *y);
printf("%p", y);
```

0x9B 0x9C 0x9D 0x9E 0x9F 0xA0

Next few weeks: Why Pointers are Great

- They allow functions to "share" data in a flexible way
- They allow dynamic data structures (*lists, trees, stacks, queues,...*)

(example of data structure that is not dynamic : array)