Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Software Systems

Lectures Week 6

Introduction to C

Pointers – String.h – Functions - Structures

Prof. Joseph Vybihal
Computer Science
McGill University

Part 1

Pointers, Strings and String.h

Pointers, Strings, and string.h

Pointers are variables that can directly reference other computer structures through the structure's address.

Strings, in C, are implemented with pointers.

The library string.h has many string manipulation functions.

Pointers

By example:

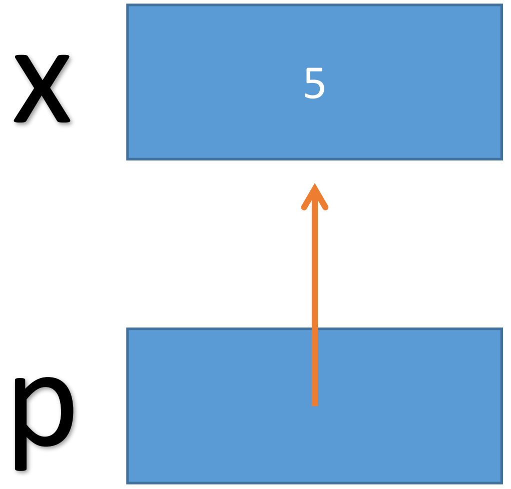
- int x = 5; // is a simple structure that stores integer numbers
- int *p;   // create a variable that can store the address of
                    another structure
- p = &x;   // p has been assigned the address of x
                    p is said to be “pointing to” x
                    p does not have the value of x, p just “knows”
                    where x is located in the computer’s memory
- printf(“%d”, x);   // prints 5 to the screen
- printf(“%d”, *p); // prints 5 to the screen

&	→	return the address of a structure (referencing)
*	→	using the address, get the value from the location in the computer’s memory (derefrencing)

Pointers Visually

By example:

- int x = 5;
- int *p;
- p = &x;
- printf(“%d”, x);
- printf(“%d”, *p);
- printf(“%d”, p);



&	creates the arrow
*	follows the arrow

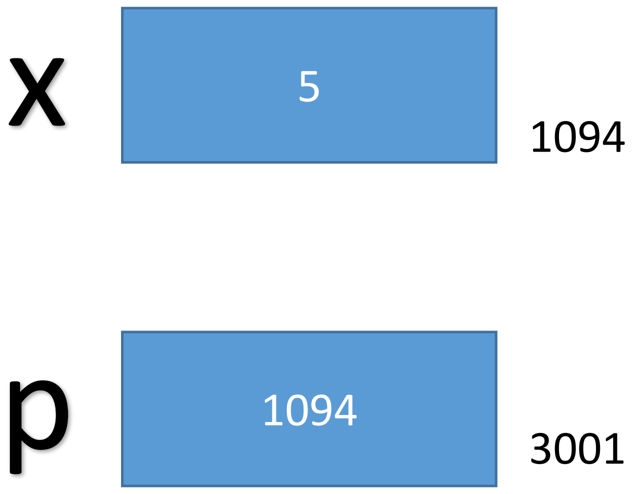
Unix
Bash
C
GNU
Systems

Pointers structurally

Pointers, in C, are stored as integer numbers.

This means we can do math with them.

```
 x = 5;
 p = &x;

 printf(“%d”, *p);
 printf(“%d”, p);

 p = p + 1;
 printf(“%d”, p);   // 1095
Printf(“%d”, *p); // ??
Printf(“%d”, &p); // 3001
Printf(“%d”,*(p-1)); // 5
```



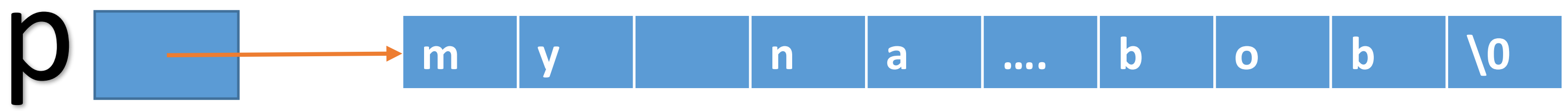
Strings

Definition:

- In C, a string is defined as a constant, a series of contiguous characters ending with a special end-of-string character called the null character, represented by '\0'.

Syntax:

- char *p = "my name is bob";
- The p is a variable pointer
  - The p points to the first character (in this case 'm')
- The "my name is bob" is a static constant value
  - It cannot be edited
  - It cannot be written over



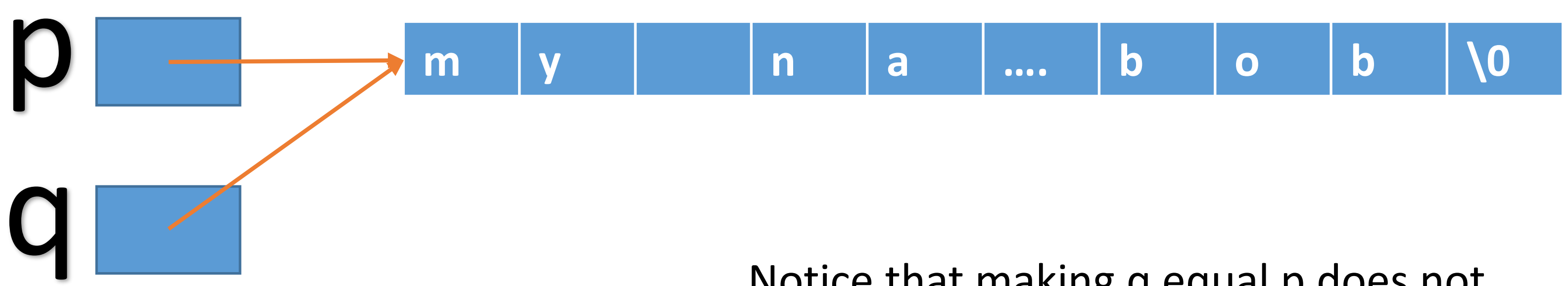
String Manipulation

```
char *p = “my name is bob”;
char *q;

printf(“%s”, p);    // outputs: my name is bob
printf(“%c”, *p); // outputs: m
printf(“%s”, *p); // outputs: … use ‘m’ as an address!! and print from there

printf(“%s”, (p+1)); // outputs: y name is bob

q = p + 3;
printf(“%s”, q); // outputs: name is bob
```

Note: The %s in printf will print character by character from the string until it comes to the \0 character. If the \0 character was missing then printif would not stop printing until it came to a \0 or crashed somewhere in memory.

String Manipulation

```
char *p = “my name is bob”;
char *q;


q = p;
```



Notice that making q equal p does not create a new copy of the string.

String Manipulation

```
char *p = “my name is bob”;
scanf(“%s”, p); // this will crash or behave strangely
```

Since p is a variable this compiles.

But p points to a constant static string.

At run-time scanf will attempt to write over the constant, which is illegal. The solution is to use an array. Arrays are not constant.

String Manipulation

```
char *a=”bob”;
char *b=”bob”;


if (a == b) // false
```

Note: a and b contain the same info
But this information is in a different location.

String.h

Important Functions

- int strcmp(char *s1, char*s2)
- int strncmp(char *s1, char*s2, int len)
- int strlen(char *string)
- char *strcpy(char *dest, char*src) … strncpy
- char *strcat(char *dest, char*src) … strncat
- void *memset(char* string, char character, int len)

The strcmp function

Calculates the difference between two strings:

- Returns 0 when equal
- Returns >0 when first argument is larger than second
- Returns <0 when first argument is smaller than second

```
char *a=”bob”;
char *b=”bob”;

if (strcmp(a,b)==0) // then it is the same
if (strcmp(a,b)!=0) // then it is not the same

if (!strcmp(a,b)) // then this is the SAME, don’t do this…
```

```
int x = strcmp(“bob”, “bob”);       //     0
int x = strcmp(“bob”,”mary”);       //   <0
int x = strcmp(“mary”,”bob”);       //   >0

int x = strncmp(“mark”,”mary”,3); //  0

int x = strlen(“mary”);             // x = 4

char array[100];
strcpy(array, “first words”);
strcat(array, “second words”);
printf(“%s”, array);                // outputs: first wordssecond words

char array2[100];
memset(array2, ‘*’, 50);            // first 50 cells are *
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

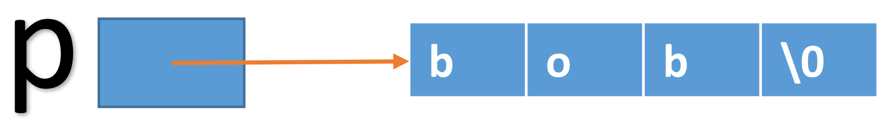
#include <string.h>

```
void    *memccpy(void *restrict, const void *restrict, int, size_t);

void    *memchr(const void *, int, size_t);
int     memcmp(const void *, const void *, size_t);
void    *memcpy(void *restrict, const void *restrict, size_t);
void    *memmove(void *, const void *, size_t);
void    *memset(void *, int, size_t);
char    *strcat(char *restrict, const char *restrict);
char    *strchr(const char *, int);  <--------------------
int     strcmp(const char *, const char *);
int     strcoll(const char *, const char *);
char    *strcpy(char *restrict, const char *restrict);
size_t  strcspn(const char *, const char *);
char    *strdup(const char *);  <--------------------
char    *strerror(int);
int     *strerror_r(int, char *, size_t);
size_t  strlen(const char *);
char    *strncat(char *restrict, const char *restrict, size_t);
int     strncmp(const char *, const char *, size_t);
char    *strncpy(char *restrict, const char *restrict, size_t);
char    *strpbrk(const char *, const char *);
char    *strrchr(const char *, int);
size_t  strspn(const char *, const char *);
char    *strstr(const char *, const char *); <--------------------
char    *strtok(char *restrict, const char *restrict);
char    *strtok_r(char *, const char *, char **);
size_t  strxfrm(char *restrict, const char *restrict, size_t);
```

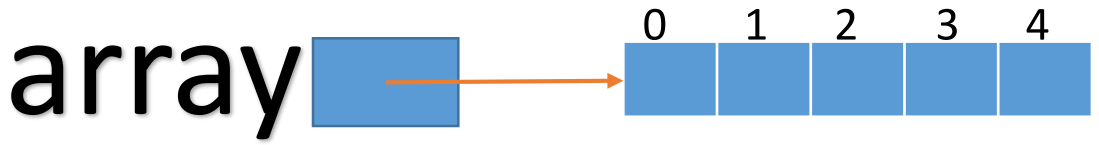
Unix
Bash
C
GNU
Systems

Arrays, Strings, and pointers

char *p = “bob”;



char array[5];



- Notice that they are structurally similar.
- This means they are interchangeable in many contexts within C.
- TYPE* and TYPE[] are interchangeable.

Example

```
char array[100];

scanf(“%s”, array);

strcat(array, “ extra stuff”);

printf(“%s”, array); // What does it print out ?
```

Example

```
char array[100];
char *p, *q;

p = array;
printf("%s", p); // what is printed out?
printf("%s", (p+2)); // what is printed out?


q = &array[5]; // since we are not pointing to the first cell
printf("%s", q); // what is printed out?
```

Implementing string.h functions using pointers.

```
int strlen(char *str)
{
        int i;

        for(i=0; *str != '\0'; i++, str++);

        return i;
}
```

```
int strcmp(char *s1, char *s2)
{
        for(; *s1!='\0' && *s2!='\0' && *s1==*s2; s1++,s2++);

        return *s1-*s2;
}
```

Example

Ask for a person's first name and last name separately.

Display an error message of they input the same world for both the first and last name.

Otherwise, concatenate the names as LAST, FIRST and display the length of the full name.

Part 2

Functions and Scope

Functions

Syntax:

- RETURN_TYPE FN_NAME (PARAMETERS) { BODY; return VAR; }

Where:

- RETURN_TYPE any legal C type declaration
  - The type void can be used to mean nothing will be returned. In this case the "return VAR;" statement is not used.
- FN_NAME the functions name, must be unique
- PARAMETERS a comma separated list of TYPE VAR, TYPE2 V2
- BODY C code
- Return VAR statement specifying what is returned, must agree with RETURN_TYPE

Example

```
int max(int a, int b, int c) {                           // Function Declaration
     int theMax = a; // we assume ‘a’ is the max

     if (b > theMax) theMax = b;
     if (c > theMax) theMax = c;

     return theMax;
}
int main() {
     int x, y, z, result;
     scanf(“%d %d %d”, &x, &y, &z);
     result = max(x,y,z);                                // Function “call” or “invocation”
     printf(“%d\n”, result);
}
```

Compilation Order

2-pass vs 1-pass compilers

- 2-pass compiler: scans the source file twice
- 1-pass compiler: scans the source file once
- Conclusion: 1-pass is faster than 2-pass
- Restriction: 1-pass, by definition, has a declaration restriction

C is a 1-pass compiler

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Compilation Order

The 1-pass declaration order restriction:

- All identifiers must be declared before they are used
- Example identifiers:
  - Variables, function names, pre-processor directives, user-defined types, libraries

```
int main() {
  x = 10;
  int x;
  x = negate(x);
}

int negate(int a) {
  return a * -1;
}
```

Fails at “int x” and function call.

```
int negate(int a) {
  return a * -1;
}

int main() {
  int x;
  x = 10;
  x = negate(x);
}
```

Fixed.

Scope

Scope defines how the compiler determines which identifier declaration is being used in a statement.

First come first server scope rule:

1. Declaration and usage is in the same Block
2. Declaration and usage is in the same Local space
3. Declaration and usage is in the same Global file space
4. Declaration and usage is in the same External space
5. Generate syntax error

Example

```
#include <stdio.h>

int x;                                  // x is in global file space

int max(int a, int b, int c) {          // a, b, c are in the “max” local space

    int theMax = a;                     // theMax is in the “max” local space

    if (b > theMax) theMax = b;

    if (c > theMax) theMax = c;

    return theMax;

}

int main() {

    int limit = 10;                     // limit is in the “main” local space

    for(int x=0; x<limit; x++) {        // x is in the “for” block space

      int result = max(x,x+1,x+2);      // result is in the “for” block space

    }

    printf(“%d”, result);

}
```

In the main program: what happens with ‘x’ and ‘result’ in terms of scope?

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Function Prototypes

```
#include <stdio.h>
int x;
int max(int,int,int);                    // this is a function prototype
int main() {
    int limit = 10;
    for(int x=0; x<limit; x++) {
      int result = max(x,x+1,x+2);
    }
    printf("%d", result);
}
int max(int a, int b, int c) {
    int theMax = a;
    if (b > theMax) theMax = b;
    if (c > theMax) theMax = c;
    return theMax;
}
```

A function prototype is a promise to the compiler.

"I promise that somewhere in this source file there is a declaration for this function. And, that function will look exactly like this prototype."

Notice that a prototype does not have a body. It simply ends with a semi-colon.

Call-by-value

Definition:

- Passing a copy of a variable to a function.

Conclusion:

- Changing the value of the variable in the function does not effect the value of the variable in the calling environment.

Example:

```
void increment (int a) {
  a = a + 1;
}

int main() {
  int x = 5;
  increment(x);     // since void was used we don't return anything
  printf("%d", x);  // even though 'a' was incremented the value of 'x' is still 5
}
```

Call-by-reference

Definition:

- Passing a pointer to the original variable to a function.

Conclusion:

- Changes to the value of the local variable will also effect the value in the original variable.

Example:

```
void increment (int *a) {
  *a = *a + 1;
}

int main() {
  int x = 5;
  increment(&x);     // sends the address of ‘a’ to the function (like in scanf!)
  printf(“%d”, x);     // this will print out 6
}
```

Call-by-reference and value

(arrays and strings)

```
int findA(char array[], int length, char key) {        // array is by reference, the others not
    int pos;
    for(pos=0; pos<length; pos++) if (array[pos] == key) return pos;
    return -1; // to indicate not in array since arrays have index numbers >=0
}
int findS(char *s, char key) {        // string is by reference (but strings are constants)
    char *p;
    for(p=s; *p!=‘\0’; p++) if (*p == key) return p-s; // like pos above, distance.
    return -1;
}
int main() {
    char name[30], address[100];
    printf(“%d”, findA(name,30,’w’));          // notice this works for different sized arrays
    printf(“%d”, findA(address,100,’5’));      //    due to the [] in the function declaration
    printf(“%d”,findS(“My name is bob”, ‘b’));
```

Question

How can we write a function called swap:

void swap(value1, value2)

So that after this function is called the values in the two original variables have been exchanged?

```
int x=5, y=10;
swap(x, y);
printf("%d %d",x,y); // 10 5
```

Question

How can we program factorial recursively?

(hint: same as in Java)

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Part 3

Struct and Union

Complex Data Structures

User-defined types composing primitive elements into a single complex structure.

Two structures exist in C:

- The <u>struct</u> data structure
- The <u>union</u> data structure

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

struct

Syntax:

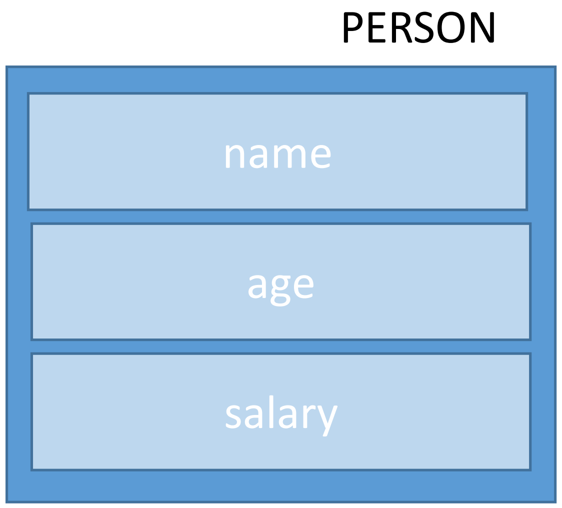
```
struct OPTIONAL_NAME {
    FIELDS;
} OPTIONAL_VAR_NAME ;
```

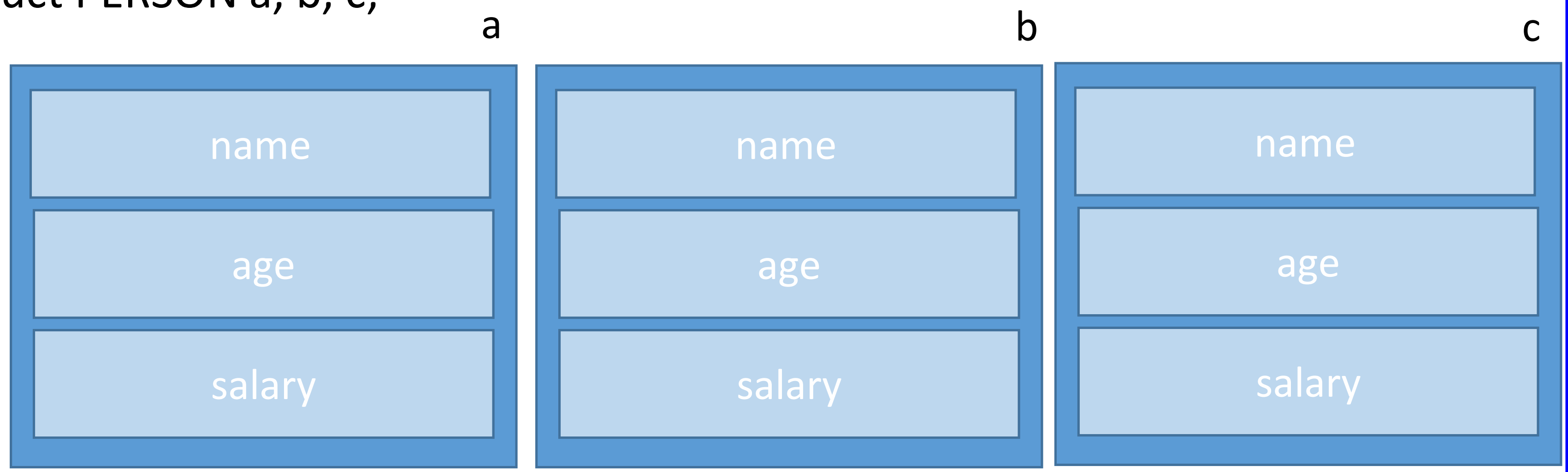
Where:

- OPTIONAL_NAME - is its user-defined type name
- OPTIONAL_VAR_NAME- is a variable containing this structure
- FIELDS
    - Is a list of semi-colon separated variable declarations
    - TYPE1 VAR1; TYPE2 VAR2; etc.

Example

```
#include <stdio.h>
struct PERSON {      // customary to write in all caps
    char name[30];
    int age;
    float salary;
};     // notice the semi-colon
int main() {
    struct PERSON a, b, c;
etc…
```





The dot operator

To access the fields within the structure we use the dot operator:

```
struct PERSON a;

scanf(“%s”, a.name);
scanf(“%d”, &(a.age));
a.salary = 50.25;

printf(“%s %d %f”, a.name, a.age, a.salary);
```

Example

```
#include <stdio.h>
struct PERSON { char name[30]; int age; float salary; } a, b, c; // 3 variables
int main() {
    printf("Enter the name for person a: ");
    scanf("%s", a.name);
    printf("Enter the age: ");
    scanf("%d", &(a.age));
    printf("Enter the salary: ");
    scanf("%f", &(a.salary));
    // repeat the above for b and c
    printf("Enter the name for person b: ");
    scanf("%s", b.name);
}
```

Array of struct

```
#include <stdio.h>
struct PERSON { char name[30]; int age; float salary; } ;
int main() {
    struct PERSON people[100];
    for(int x=0; x<100; x++) {
            scanf(“%s”, people[x].name);
            scanf(“%d”, &(people[x].age));
            scanf(“%f”, &(people[x].salary));

            if (people[x].age == 20) printf(“Hey you are 20!!\n”);
    }
}
```

union

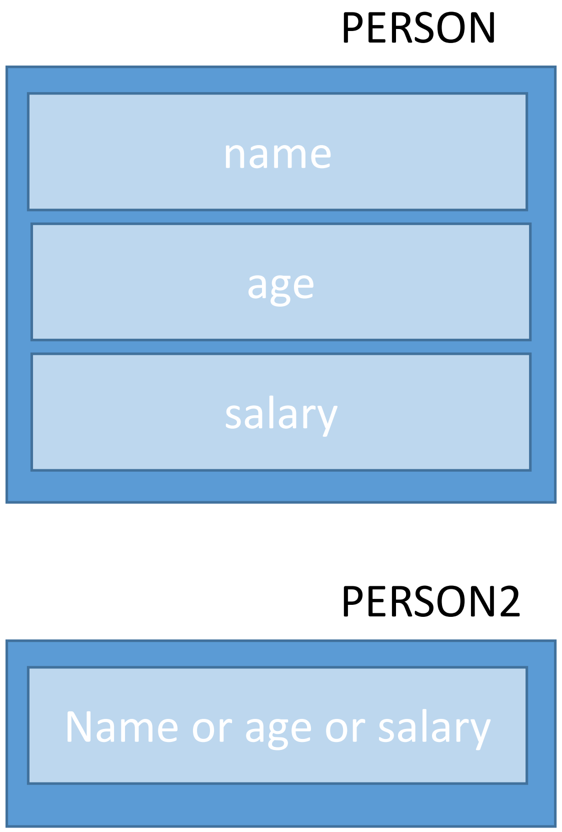
Syntax:

```
union OPTIONAL_NAME {
    FIELDS;
} OPTIONAL_VAR_NAME ;
```

It is similar to struct in syntax, but not in what it builds.

Example

```
struct PERSON {
    char name[30];
    int age;
    float salary;
};


union PERSON2 {
    char name[30];
    int age;
    float salary;
};
```



```
union PERSON2 x;
x.age = 20;
x.salary = 30.7; // destroys the 20
```

Example

```
struct PERSON {
    char type;   // ‘p’=prof, ‘s’=student, ‘f’=staff
    char name[30];
    int age;
    char ID[10];
    union SPECIFIC_DATA {
            struct {
                    float evaluation;
                    int position;
            } prof;
            struct { float GPA; float fees; } stud;
            struct { char level; float salary; } staff;
    } specific;
}  people[100];
```

Example

```
struct PERSON mcgill[1000];

int findPerson(char ID[]) {
    int pos;
    for (pos=0; pos<1000; pos++) {
            if (strcmp(mcgill[pos].ID, ID)==0) return pos;
    }
    return -1;
}

int main() {
    int location = findPerson(“3219678”);
    if (location != -1) printf(“%s”, mcgill[location].name);
```

Example

```
struct PERSON mcgill[1000];
int main() {
    int location = findPerson(“3219678”);
    if (location != -1) {
            printf(“%s”, mcgill[location].name);
            switch (mcgill [location].type ) {
                case ‘p’:
                    printf(“Evaluation= %f\n”, mcgill[location].specific.prof.evaluation);
                    break;
                case ‘s’:
                    printf(“GPA = %f\n”, mcgill[location].specific.stud.GPA);
            }
    }
}
```

Question

If a bank has a checking account and a savings account. How might we build a struct and union data structure to represent this information?

CHECKING	SAVINGS
Account number	Account number
Balance	Balance
	Withdraw fee

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Question

How is this similar to polymorphism?