Computer programming

"Repetitio est mater studiorum"

Latin proverb

C Pointers

Review

Key points

- Pointers and pointer operators.
- Pass arguments to functions by reference by using pointer.
- sizeof Operator and Pointer Arithmetic.
- The close relationships among pointers, arrays and strings.
- Pointers to functions.

Pointers and pointer operators

- Pointer:
  - Contains address of a variable that has a specific value (indirect reference)
  - Definitions: `int *myPtr1, *myPtr2;`
- Operators:
  - Address operator: &
  - Indirection/dereferencing operator: *
  - * and & are inverses, They cancel each other out

```
int y = 5;
int *yPtr;
yPtr = &y;  /* yPtr gets address of y */
*yptr = 7;  /* changes y to 7 *
```

Pass arguments to functions by reference by using pointer.

- Pass address of argument using & operator
  - Arrays are not passed with &
- Used as alias/nickname for variable inside of function

```
/*Function definition*/
void double( int *number )
 {
   *number = 2 * ( *number );
 }
int main (void) {
   int a =5;
   /*Function call*/
   double( &a )
   printf(“%d /n”, a)
}
```

  - Const qualifiers: variable can not be changed

```
int *const myPtr = &x;    /*Constant pointer*/
const int *myPtr = &x;    /*Constant int*/
const int *const Ptr = &x; /*Cons. pointer point to Cons. int*/
```

sizeof Operator and Pointer Arithmetic.

- `Sizeof:` Returns size of operand in bytes
  - Can be use for program that depend on data type sizes and run on different system
- Pointer Arithmetic:
  - ++, --, +,+=,-,-=
  - Operations meaningless unless performed on an array
  - Unit: base on array type and system

```
int v[10], *vPtr2, *vPtr;
vPtr2 = &v[ 2 ];
vPtr = &v[ 0 ];
vPtr2 – vPtr = 2;
vPtr = 3000;
vPtr += 2; /*vPtr = vPtr2  = 3008; 1 unit = 4
  bytes*/
```

The close relationships among pointers, arrays and strings.

- Arrays and pointers closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations
- Equalities between pointer bPtr and array b[] (assume bPtr is a pointer to type of array b)
  - `bPtr = b;`
  - `bPtr = &b[0];`
  - `bPtr[3] = b[3];`
  - `*( bPtr + 3 ) = b[3];`
- Arrays of pointers

  -
    ```
    char *suit[ 4 ] = { "Hearts", "Diamonds",
     "Clubs", "Spades" };
    ```

Pointers to functions

- Function pointers
  - Contains address of function
  - Function pointers can be
    - Passed to functions
    - Stored in arrays
    - Assigned to other function pointers
- Two definition ways:
  - `int *compare( int a, int b )`
    - `Returns a pointer to int (not function pointer)`
  - `int ( *compare )( int a, int b )`
    - `Returns an int`

Pointers to functions

- Returning function pointers
- Arrays of function pointers
- ... (see Lecture 8; study examples)

C Characters and Strings

Review

Key points

- Fundamental of Strings and Character.
- Character-handling library (ctype).
- String-conversion functions (stdlib).
- Standard Input/Output library functions(stdio).
- String-processing functions of the String-Handling Library (string).
- Memory functions of the the String-Handling Library.

Fundamentals of Strings and Characters

- Characters
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An `int` value represented as a character in single quotes
    - `'z'` represents the integer value of `z`
- Strings
  - Strings are arrays of characters
    - String is a pointer to first character
    - Value of string is the address of first character
  - String definitions

```
char color[] = "blue";
char *colorPtr = "blue";
```

  - Inputting strings
    - `scanf("%s", word);`
  - Remember that strings represented as character arrays end with '\0'

Character Handling Library

- #include <ctype.h>

Prototype	Function description
`int toupper( int c );`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns a true value if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 otherwise.
`int iscntrl( int c );`	Returns a true value if `c` is a control character and 0 otherwise.
`int ispunct( int c );`	Returns a true value if `c` is a printing character other than a space, a digit, or a letter and returns 0 otherwise.
`int isprint( int c );`	Returns a true value if `c` is a printing character including a space (`' '`) and returns 0 otherwise.
`int isgraph( int c );`	Returns a true value if `c` is a printing character other than a space (`' '`) and returns 0 otherwise.

Character Handling Library

Prototype	Function description
`int isdigit( int c );`	Returns a true value if `c` is a digit and 0 (false) otherwise.
`int isalpha( int c );`	Returns a true value if `c` is a letter and 0 otherwise.
`int isalnum( int c );`	Returns a true value if `c` is a digit or a letter and 0 otherwise.
`int isxdigit( int c );`	Returns a true value if `c` is a hexadecimal digit character and 0 otherwise. (See Appendix E, Number Systems, for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower( int c );`	Returns a true value if `c` is a lowercase letter and 0 otherwise.
`int isupper( int c );`	Returns a true value if `c` is an uppercase letter and 0 otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.

String-Conversion Functions

- #include `<stdlib.h>`

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`double strtod( const char *nPtr, char **endPtr );`	Converts the string `nPtr` to `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `unsigned long`.

Standard Input/Output Library Functions #include <stdio.h>

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`int putchar( int c );`	Prints the character stored in `c` and returns it as an integer.
`int puts( const char *s );`	Prints the string `s` followed by a newline character. Returns a non-zero integer if successful, or `EOF` if an error occurs.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs.
`int sscanf( char *s, const char *format, ... );`	Equivalent to `scanf`, except the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs.

String Manipulation Functions of the String Handling Library

- #include <string.h>

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most `n` characters of string `s2` into array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most `n` characters of string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

Comparison Functions of the String-Handling Library

Function prototype	Function description
`int strcmp( const char *s1, const char *s2 );`	Compares the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to `n` characters of the string `s1` with the string `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`, respectively.

Search functions of the String-Handling Library

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting of characters not contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.

Search functions of the String-Handling Library

Function prototype	Function description
`char *strrchr( const char *s, int c );`	
	Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	
	Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );`	
	A sequence of calls to `strtok` breaks string `s1` into “tokens”—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

Memory Functions of the String-Handling Library

Function prototype	Function description
`void *memcpy( void *s1, const void *s2, size_t n );`	
	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. A pointer to the resulting object is returned.
`void *memmove( void *s1, const void *s2, size_t n );`	
	Copies `n` characters from the object pointed to by `s2` into the object pointed to by `s1`. The copy is performed as if the characters were first copied from the object pointed to by `s2` into a temporary array and then from the temporary array into the object pointed to by `s1`. A pointer to the resulting object is returned.
`int memcmp( const void *s1, const void *s2, size_t n );`	
	Compares the first `n` characters of the objects pointed to by `s1` and `s2`. The function returns `0`, less than `0` or greater than `0` if `s1` is equal to, less than or greater than `s2`.
`void *memchr( const void *s, int c, size_t n );`	
	Locates the first occurrence of `c` (converted to `unsigned char`) in the first `n` characters of the object pointed to by `s`. If `c` is found, a pointer to `c` in the object is returned. Otherwise, `NULL` is returned.
`void *memset( void *s, int c, size_t n );`	
	Copies `c` (converted to `unsigned char`) into the first `n` characters of the object pointed to by `s`. A pointer to the result is returned.

Other functions of the string-handling library

Function prototype	Function description
`char *strerror( int errornum );`	
	Maps `errornum` into a full text string in a locale-specific manner (e.g. the message may appear in different languages based on its location). A pointer to the string is returned.
`size_t strlen( const char *s );`	
	Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

C Formatted Input / Output

Review

Key points

- Input and output streams.
- Format output using `printf`.
- Format input using `scanf`.
- What header do we include?

Input and output streams.

- Streams
  - Sequences of characters organized into lines
  - Performs all input and output
  - Can often be redirected
- Output:
  - `printf, sprint, puts, putchar`
- Input:
  - `scanf, sscanf, gets, getchar`

Format output using printf

- `printf(` *format-control-string, other-arguments* `);`
  - printf(**"\t%.3f\n\t%.3e\n\t%.3g\n\n"**, f, f, f);
- Conversion specifiers
  - Integer: d, i, o, u, x, X, h, l
  - Floating point: f, e, E, g, G, L
  - Character and String: c, s
  - Others: p, n, %
- Print with field widths and precision
  - Field widths: Size of field in which data is printed
    - Syntax: printf("%4d\n", -12);
  - Precision: Integer, Floating point, String
    - Syntax: printf(**"\t%.4d\n\t%.9d\n\n"**, i, i);
- Flags: -, +, space, #, 0
- Escape sequence: \

Format input using `scanf`

- `scanf(`*format-control-string, other-arguments)*
  - scanf(**"%d%i%i%i%o"**, &a, &b, &c, &d, &e,);
- Format control string:
  - Contain conversion specifiers only
  - Conversion specifiers:
    - Integers, Floating point, Character and String
    - Scan set [scan characters]

C Structures, Unions, Bit Manipulations and Enumerations

Review

Key points

- **Understanding & using**
  - **Structures**
  - **Unions**
  - **Enumerations.**
- **Bitwise operators.**

Structure

- Structure:
  - collection of related variables

```
struct card {
   char *face;
   char *suit;
 };
```

- Definition: 2 ways

```
card oneCard, deck[ 52 ], *cPtr;
```

  - Or:

```
struct card {
   char *face;
   char *suit;
} oneCard, deck[ 52 ], *cPtr;
```

  - Accessing structure members
  - (.) used with structure variables
  - (->) used with pointers to structure variables
- typedef: create shorter type names

```
typedef struct Card *CardPtr;
```

Unions

- `Union: like structure but`
  - Only contains one data member at a time
  - Members of a `union` share space
  - Only the last data member defined can be accessed
- `union` definitions
  - Same as struct

```
union Number {
   int x;
   float y;
};
union Number value;
```

Enumeration

- Enumeration
  - Set of integer constants represented by identifiers
  - Values are automatically set
  - Example:

```
enum Months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG,
  SEP, OCT, NOV, DEC};
```

    - Creates a new type `enum` Months in which the identifiers are set to the integers 1 to 12
  - Enumeration variables can only assume their enumeration constant values (not the integer representations)

10.9 Bitwise Operators

Bit 1	Bit 2	Bit 1 & Bit 2
0	0	0
1	0	0
0	1	0
1	1	1

Bit 1	Bit 2	Bit 1 | Bit 2
0	0	0
1	0	1
0	1	1
1	1	1

Bit 1	Bit 2	Bit 1 ^ Bit 2
0	0	0
1	0	1
0	1	1
1	1	0

Bitwise assignment operators	
&=	Bitwise AND assignment operator.
|=	Bitwise inclusive OR assignment operator.
^=	Bitwise exclusive OR assignment operator.
<<=	Left-shift assignment operator.
>>=	Right-shift assignment operator.

Bit Fields

- Bit field : Enable better memory utilization

```
struct BitCard {
   unsigned face : 4;
   unsigned suit : 2;
   unsigned color : 1;
};
```

  - Must be defined as `int` or `unsigned`
  - Cannot access individual bits
  - Member of a structure whose size (in bits) has been specified

C File Processing

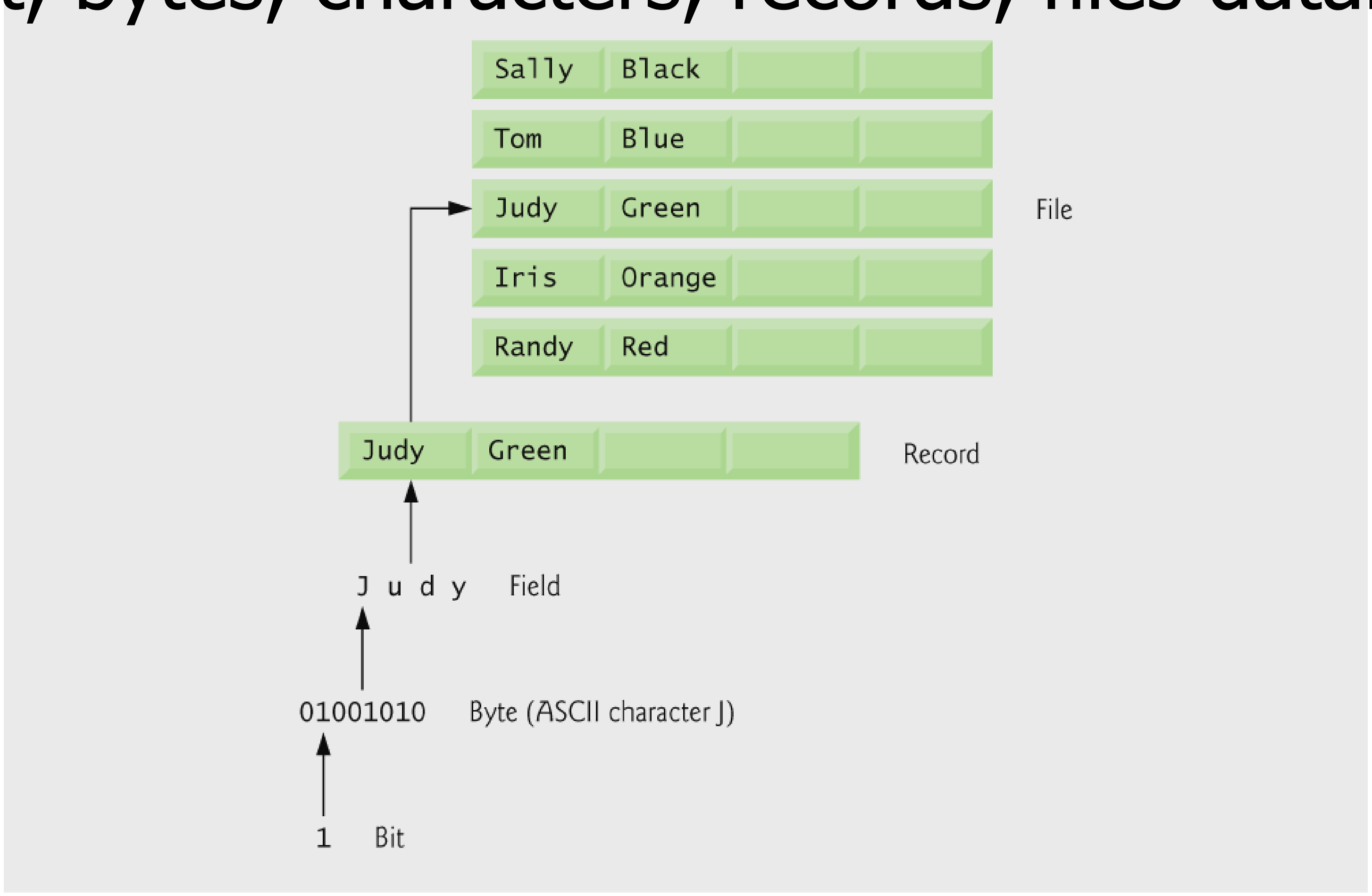
Review

Key points

- Data hierarchy
- File & Stream
- Create, read, write and update:
  - Sequential access file
  - Random access file

Data hierarchy

- Bit, bytes, characters, records, files database



File & Stream

- File:
  - A sequence of bytes, ends with *end-of-file marker*
  - Structure
    - File descriptor
    - File control block
- Stream:
  - Provide communication channel between files and programs
  - File pointer: to be returned when open a file
    - stdin, stdout, stderr
- Read/Write function
  - fgetc, fputc, fgets, fputs, fscanf, fprintf
- What header file do we include?

Create, read, write and update

- Create

```
FILE *cfPtr;
cfPtr = fopen(“clients.dat", “w”);
```

  - Opening modes
    - Character mode: r,w,a,r+,w+,a+,
    - Binary mode: rb,wb,ab,rb+,wb+,ab+

Functions	Random	Sequential
Read	**fread**	**fscanf**
Write	**fwrite**	**fprintf**
Pointer position	**fseek**	**rewind**
Others	**feof, fclose**	

Sequential vs. Random access file

Properties	Random	Sequential
Length of records	Fixed	Dynamic
Access individual records without searching through other records	Yes	No, should be searched through other records
Instant access to records	Yes	No
Data can be inserted without destroying other data	Yes	No, Other data can be destroyed
Data previously stored can be updated or deleted without overwriting	Yes	No
Storage memory	High	Low
Human readable	No	Yes

C Data Structures

Review

Key points

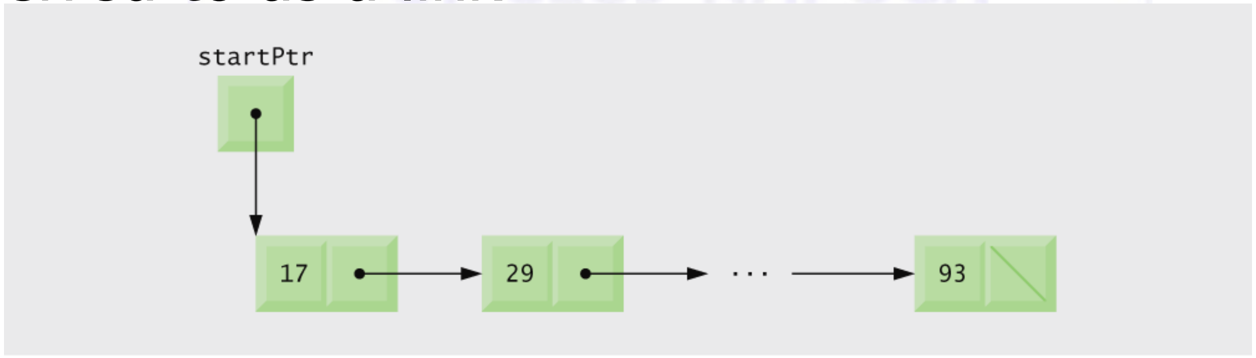
- Self-Referential Structure
- Dynamic Memory Allocation
- Dynamic data structures
  - Linked lists
  - Stacks
  - Queues

Self-Referential Structures

- Self-referential structures:
  - Structure contains a pointer to a structure of the same type

```
struct node {
   int data;
   struct node *nextPtr;
}
```

- `nextPtr`
  - Points to an object of type `node`
  - Referred to as a link



Dynamic allocation of memory Principles

- You control the duration of a dynamically-allocated object.
- You must obtain the dynamically-allocated memory from the system by using the malloc() (or calloc() or realloc()) routine.
- You must return the dynamically-allocated memory to the system by using the free() routine.
- Guard against:
  - dereferencing a null pointer
  - going outside the allocated memory block
  - freeing memory using an address that was not provided by malloc(), calloc(), or realloc()
  - leaving a dangling pointer
  - causing a storage leak

Dynamic Memory Allocation

- Dynamic memory allocation
  - `malloc ():` Obtain memory for a new structure
    - Takes number of bytes to allocate
    - Return a void * pointer or Null
    - Example: `newPtr = malloc( sizeof( struct node ) );`
  - `free ():` Release memory
    - Deallocates memory allocated by `malloc`
    - Takes a pointer as an argument
    - `free ( newPtr );`

Dynamic data structures

- Dynamic data structures
  - Data structures that grow and shrink during execution
- Linked lists
  - Allow insertions and removals anywhere
  - Can be used when the number of data element is unpredictable
- Stacks
  - Allow insertions and removals only at top of stack
  - Rule: Last in first out
- Queues
  - Allow insertions at the back and removals from the front
  - Rule: First in, first out

About the exam

- Who can attend?
  - students who got a mark >= 5 for the laboratory test
- Closed book exam
- Structure
  - theory test questions (multiple choice, fill in the blanks, ***unanswered***) – max. 1 hour
  - problem(s): develop algorithm, write C program – max. 1 hour and 45 min
- Passing condition:
  - lab. test >=5, test questions >=5, problem(s) >=5
- Grade formula: 0.35*LT + 0.65*WE

Exam preparation

- Review supplied materials
  - lecture handouts
  - laboratory guides
  - provided supplementary material (aka CD_CP)
- Probe C online tests. e.g.
  - http://gd.tuwien.ac.at/languages/c/programming-bbrown/onlinet.htm
- Sample questions for final are posted on course web site

- Success!