COMP1038 AE1PGA 21-22

Review

Jiawei Li (Michael)

TOPICS

- Structured programming (instructions + data)
- Data (types + variables)
- Program Control (instructions)
- Functions
- Arrays
- Pointers
- Memory management
- Strings
- I/O + Files
- Data structures
- Compilation knowledge + command line arguments

Keywords in C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	static	void
default	goto	sizeof	volatile
do	if	signed	while

CODE + DATA

A program can be seen as code + data. It contains a list of ingredients (called variables) and a list of directions (called statements) that tell the computer what to do with the variables. The variables can represent numeric data, text, or graphical images

INPUT & OUTPUT

Input/Output (I/O) describes any operation, program, or device that transfers data to or from a computer. Typical I/O devices are printers, hard disks, keyboards, and mouses.

VARIABLES

Variables are the abstraction used to store values in the machine.

Language design questions:

- Can their value be changed after the first time?
- Can they be used through the entire program or only in certain places?
- Where these value should be stored in memory?

- Const
- Global vs local
- Stack, heap, or data section

SELECTION

How can a program make choices?

- if, else
- switch
- goto

ITERATION & RECURSION

How can a program do the same thing a number of times?

- while
- do, while
- for

ARRAYS

- An *array* is a continuous block of variables of the same type.
- You can refer to the array as a whole by using the variable name of the array.
- You can refer to the individual elements of the array by using an *index* into the array using square brackets `[ ]`
- Array indicies start at 0 and end at *length - 1*.
- Arrays are fixed length; you cannot change the length of them after they have been created.

STRINGS

- *Strings* are how we represent runs of text in C.
- A string is just an array of `char` variables, terminated with a `'\0'` character.
- Instead of assigning each character individually, C lets you give C literals between double-quotes.
- `char name[] = "Paul";`
- This will create a char array:
- `{ 'P', 'a', 'u', 'l', '\0' }`
- The array is length *5*, the 4 characters plus the *nul* character.

POINTER VARIABLES

- For every type of variable, we also have a type to store a *pointer* to that type.
- To declare a variable of this type, put a `*` before the variable name.

```
// normal integer initialised to value 5
int x = 5;
// declare a pointer to an integer variable
int *px;
// set the pointer value to the address of the
x variable
px = &x;
```

NULL POINTER

- `NULL` pointer is used as a pointer value to indicate the pointer variable does not currently point to anything.
- Defined in `stdio.h`
- It is actually the value zero but you should always use `NULL`.

```
int *px = NULL;
// ...
// do some things that may or may not
// make px point to a variable.
// ...
if(px != NULL)
{
   printf("%d\n", *px);
}
```

COMMAND-LINE ARGUMENTS

- `int main(int argc, char *argv[])`
- `argc` is the number of command-line arguments our program was given.
- `argv` is an array of *char pointers*, therefore an array of strings.
- Each string is one of the command line arguments.

Stack and heap

- Local variables and function parameters are allocated on the stack. This is automatically done by the C compiler.
- The stack is fully controlled by the operating system.
- Dynamic memory allocation on the heap needs to be done manually.

Functions

A function is a block of code that performs a specific task. The compiler allocates memory (i.e. stack) to store a function's parameters and the variables when the function is called.

- Return value.
- Arguments.
- Function body.
- Declaration.

```
int func(int arg1, char arg2)
{
        int X;
        …

        return X;
}
```

Pass by values v.s. pass by references

- By calling by values, copies of arguments are passed to the function so data at the caller side will be unchanged.
- By calling by references, addresses of arguments are passed to the function so data at the caller side can be changed.

```
int func1(int arg1, int arg2)
{
        arg1 = arg1 + arg2;
        return arg1;
}
```

```
int func2(int *arg1, int *arg2)
{
        *arg1 = *arg1 + *arg2;
        return 0;
}
```

STRUCTURES

Structures allow us to create new custom, composite data-types in our programs.

- Use the new type anywhere you could use a basic type.
- You can have pointers to structures.
- You can have arrays of structures.
- You can pass them to and return them from functions.

- A structure cannot contain an instance of itself (because it could never actually be created).
- Structures *can* contain pointers to their own type.
- Common use is for *next* or *previous* pointers in dynamic data structures.

Main data structures

- Lists:
  - Lists are linear data structures which store elements of the list one after another.
  - Unlike arrays, we can add/remove elements without having to re-create the entire data structure.
  - Insert/remove elements from anywhere in the list.
  - Access elements anywhere in the list, but slower than arrays.
- Stacks
- Queues
- Trees
- Graphs

File

- `fopen` function opens a file and returns a *file handle* for that file.
- The file handle is used in all other file operations, so the function knows which file to operate on.
- This means you can have multiple files open at the same time because each will have it's own file handle.
- `FILE *fp = fopen("highscores.txt", "r");`
- This opens the file "highscores.txt" (in the current directory) for reading only ("r").
- `fopen` returns a pointer to a `FILE` type — we don't care what this is, we just use it as documented.

ASSESSMENT

75% COURSEWORK

3 assessed elements

- 15% 3 in-lab small programming assignments
- 25% Single larger programming assignment
- 35% Single larger programming assignment

25% EXAM

•1 hour

20-21 Past Exam paper