CSCI 2132
Software Development

Lecture 35:

Shell Scripting

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lectures

- Examples with writing and reading a binary file
- Shell Scripting: introduction, a basic example
- variables
- arithmetic operations, conditional expressions
- ‘if’ statement, ‘for’ loop
- ‘case’ statement
- **Previous Lecture**
- **Final Exam Review**

Case Statement Example

- Example:

```
#!/bin/bash
day=`date | cut -f1 -d" "`

case "$day" in
  Mon|Wed|Fri)
    echo 2132 lectures
    ;;
  Tue|Thu)
    echo no 2132 lectures
    ;;
  Sat|Sun)
    echo No lectures
    ;;
esac
```

Conditional Expression for Files

The following conditional expressions can be used on files:

`[ -e file ]` — true if file exists

`[ -f file ]` — true if file exists and is a regular file

`[ -d file ]` — true if file exists and is a directory file

`[ -r file ]` — true if file exists and is readable by the current user

`[ -w file ]` — true if file exists and is writable by the current user

`[ -x file ]` — true if file exists and is executable by the current user

There has to be a space after `[` and before `]`.

Exit Codes

- We can use `exit` command from a script
- Exit can take a numeric argument
  - `0` is for normal exit
- If no value is provided, `exit` return exit code of the last command
- Or `0` if no command was executed

Example: A backup script

- A script that takes two arguments: a source and a destination directory
- Each file from the source directory is copied to the destination directory
- Copies only regular files, and only if it does not exist in the destination directory
- Prints the file names being copied

```
#!/bin/bash
if [ ! -d $1 ]; then
  echo Source directory does not exist
  exit 1
elif [ ! -d $2 ]; then
  echo Destination directory does not exist
  exit 1
fi

for filename in `ls $1`
do
  if [ -f $1/$filename ]; then
    if [ ! -e $2/$filename ]; then
      cp $1/$filename $2/$filename
      echo $filename
    fi
  fi
done
```

Additional Examples: Dynamically Allocated Arrays

- We focus on dynamic array-based structures
- First example: Strings
- Important to remember to allocated $+1$ character for the null character
- Can use the standard C library functions for strings

Example: `concat`

- Let us consider an example:
- Implement a function `concat` which takes two strings as arguments and concatenates them
- Unlike `strcat`, the function `concat` will not change any original strings, but create a new dynamically allocated string (we need to remember to `free` it later)

```
char* concat(const char *s1, const char *s2) {
  char *result;

  result = malloc(strlen(s1) + strlen(s2) + 1);
  if (result == NULL) {
    printf("Error: malloc failed in concat\n");
    exit(EXIT_FAILURE);
  }

  strcpy(result, s1);
  strcat(result, s2);
  return result;
}
```

- Usage:

```
char *p;
p = concat("abc", "defg");
...
free(p);
```

Another String Example

- Write a C program to reverse words in a string
- A word is any sequence of non-white-space characters
- Solution approach:
  - Scan the string backward
  - Copy words to a temporary buffer
  - Copy back buffer to the string
- Fill-in-the blanks code:
  `reversewords.c-blanks`

Reversing the Words, Revisited

- Another idea:
  - Reverse the complete string
  - Reverse each word within string once more
- Fill-in-the-blanks code available in

  `~prof2132/public/reversewords2.c-blanks`

  (in one line)

Dynamically Allocated Arrays

- Similarly to strings we can allocate arbitrary arrays; for example:

```
int *array, i;
array = (int*) malloc(n * sizeof(int));
if (array == NULL) {
  ...
}
for (i = 0; i < n; i++)
  array[i] = 0;
...
free(array);
```

- or simpler

```
int *array;
array = (int*) calloc(n, sizeof(int));
if (array == NULL) {  ... /* error */ }
```

Dynamically Allocated Arrays and VLAs

- Using dynamic memory vs VLA
- VLAs are allocated and deallocated more efficiently (on stack)
- Heap more appropriate for large arrays
- Heap appropriate for arbitrary lifespan
- Heap more appropriate for portability (C99)

Mergesort Using Dynamic Arrays

- Fill-in-the-blanks code available at:

  `~prof2132/public/mergesort3.c-blanks`

Dynamic Arrays: Resizable Arrays

- How to implement something like ArrayList in Java, or vector class in C++?
- Pseudocode for adding elements:

```
If array is full
   Resize the array to twice its current capacity
       using realloc
   Store the new element
```

- The main structure:

```
struct vector {
   int *array;
   int capacity;
   int size;
}
```

Resizable Array Implementation:

`dynamicarray.c-blanks`

- Fill-in-the-blanks implementation can be found at:

  `~prof2132/public/dynamicarray.c-blanks`

  (one line)

Dynamic Array: Time Complexity

- What can happend when `push_back` is called regarding execution time?
- Why do we consider this a reasonably efficient solution?
- Why we must be careful when determining when to shrink the array?
- This provides a better explanation of behaviour of the ArrayList in Java and the vector class in C++