Din Bostandzic
CSE 13S

Assignment 7 - Lempel Ziv Compression
Design Document

The program is the implementation of a trie which is used to store codes, each corresponding to a certain symbol, which is thus used by a word table structure to reference these symbols and to get their codes in a fast access manner. Files and their permissions are opened and manipulated using system calls, in blocks of 4kb. The program is 2 programs in one, such that there is a compression and decompression. The compression will read in a file and then write out to buffer and outfile its contents, using variable bit length encoding to reduce file size and allow decompression to occur. The decompression similarly, opens the compressed file and reads in data to place into buffer, which is then used for the word table to translate what each codes represent.

The following functional decomposition has been used along with the respective pseudocode below:

3.0 Lempel Ziv - Encode
- 3.1 bitlen(in code as unsigned 16 bit integer)
- 3.2 compress(in infile as integer, in outfile as integer)

3.1 Lempel Ziv - Decode
- 3.3 bitlen(in code as unsigned 16 bit integer)
- 3.4 decompress(in infile as integer, in outfile as integer)

<u>trie Design</u>

```
Begin trie_node_create(in code as unsigned 16 bit integer)
        Dynamically create a trenode of type structure TrieNode
        Begin if (treenode == NULL)
                return NULL
        End if
        Begin for (integer i = 0, i < ALPHABET, i++)
                treenode->children[i] = NULL
        End for
        treenode->code = code
        return treenode
End trie_node_create

Begin trie_node_delete(in n as pointer to TrieNode)
        Deallocate dynamic memory of n
        n = NULL
End trie_node_delete
```

```
Begin trie_create()
        Declare root_node as pointer to TrieNode, assign it trie_node_create(EMPTY_CODE)
        return root_node
End trie_create()

Begin trie_reset(in root as TrieNode pointer)
        Declare next as TrieNode pointer with value root
        Begin for (integer i = 0, i < ALPHABET, i++)
                TrieNode *curr = root->children[i]
                Begin if (curr->children[i] == NULL)
                        curr->code = NULL
                End if
                Begin while (curr->code != NULL)
                        Begin if (curr->children[i]->code != NULL)
                                next = curr->children[i]
                                curr->code = NULL
                                curr = next
                        End if
                        Begin if (curr->children[i] == NULL)
                                curr->code = NULL
                        End if
                End while
        End for
End trie_reset

Begin trie_delete(in n as TrieNode pointer)

        TrieNode *next = n
        Begin for (integer i = 0, i < ALPHABET, i++)
                TrieNode *curr = n->children[i]
                Begin if (curr->children[i] == NULL)
                        curr->code = NULL
                        trie_node_delete(curr)
                End if

        Begin while (curr != NULL)
                next = curr->children[i]

                curr->code = NULL
                trie_node_delete(curr)
```

```
            curr = next
            Begin if (curr == NULL)
                  trie_node_delete(curr)
            End if
      End while
  End for
End trie_delete


Begin trie_step(in n as TrieNode pointer, in sym as unsigned 8 bit integer)

      TrieNode *curr = n->children[sym]
      Begin if (curr == NULL)
            return NULL
      End if
      Begin if (curr != NULL)
            return curr
      End if
      return n
End trie_step
```

word design

```
Begin word_create(in syms as pointer to unsigned 8 bits, in len as unsigned 64 bits type)
      Dynamically create a new word
      Check if its NULL, return Null
      wrd->len = len
      Dynamically create an array syms of type unsigned 8 bits, length of len
      Check if NULL, return NULL
      return wrd
End word_create

Begin word_append_sym(in w as pointer to Word, in sym as unsigned 8 bits integer)
      Word *rw = NULL
      Begin if (w != NULL)
            Dynamically reallocate the syms array to increase len by 1
            w->syms[w->len] = sym
            rw = word_create(w->syms, w->len)
      End if
```

```
        Begin if (w != NULL)
                Begin if (w->len == 0)
                        Dynamically create an array syms of type unsigned 8 bits, length of len
                        rw->syms[0] = sym
                End if
        End if
        return rw
End word_append_sym

Begin word_delete(in w as pointer to W)
         Deallocate w
         w = NULL
End word_delete

Begin wt_create()
        Dynamically create MAX_CODE-1 wt of type WordTable
        Word *word = word_create(0, 0)
        wt[EMPTY_CODE] = word
        return wt
End wt_create

Begin wt_reset(in wt as pointer to W)
        Begin for (unsigned integer i = 2, i < MAX_CODE - 1, i++)
                Begin if (wt[i] != NULL)
                         wt[i]->syms = 0
                         wt[i]->len = 0
                End if
        End for
End wt_reset

 Begin wt_delete(in wt as pointer to WordTable)
        word_delete(wt[EMPTY_CODE])

        Begin for (unsigned integer = 2, i < UINT16_MAX - 1, i++)
                Begin if (wt[i] != NULL)
                        word_delete(wt[i])
                End if
        End for
        Deallocate wt
        wt = NULL
```

End wt_delete

io data design

```
Declare as static integer offset = 0
Declare  as static integer wrtoffset = 0
Declare  as static 8 bit integer readbuf[4096]
Declare as static 8 bit integer writebuf[4096]
Declare as static integer bits_written = 0
Declare as static integer bits_read = 0

Begin read_bytes(in infile as integer, in buf as pointer to unsigned integer 8 bit, in to_read as integer)
        Declare integer read_bytes
        Begin while ((read_bytes = read(infile, buf + offset, to_read)) != 0)
                offset += read_bytes
        End while

        Begin if (offset == to_read)
                return to_read
        End if

        return offset
End read_bytes

Begin write_bytes(in outfile as integer, in buf as pointer to unsigned 8bit integer, in to_write as integer)
        Declare integer write_bytes
        to_write = offset

        Begin while ((write_bytes = write(outfile, buf + wrtoffset, to_write - wrtoffset)) > 0)
                wrtoffset += write_bytes
        End while
        Begin if (wrtoffset == to_write)
                return to_write
        End if
        return wrtoffset
End write_bytes


Begin read_header(in infile as integer, in header as FileHeader pointer)
        Begin if (header->magic == MAGIC)
                read_bytes(infile, (uint8_t *)header, sizeof(FileHeader))
```

```
        End if
End read_header


Begin write_header(in outfile as integer, in header as FileHeader pointer)

        write(outfile, (uint8_t *)header, sizeof(FileHeader))
        offset += sizeof(FileHeader)
End write_header

Begin read_sym(int infile, uint8_t *sym)

        Declare toread as boolean = 1
        Declare static integer i = 8
        Declare static integer readinbytes = 0

        Begin if (i == 8)
                readinbytes = read_bytes(infile, readbuf, sizeof(readbuf))
        End if
        Begin if (readinbytes < 4096)
                *sym = readbuf[i]
                ++i
        Begin if (i == readinbytes)
                toread = 0
        End if
        End if
  return toread
End read_sym


buffer_pair(in integer outfile, in unsigned 16bit integer code, in unsigned 8bit integer sym,
            in unsigned 8bit integer bitlen)

        Begin for (integer i = 0, i < bitlen, i++) {
                Begin If (bits_written == (offset)*8)
                        write_bytes(outfile, writebuf, sizeof(writebuf))
                        bits_written = 0
                End if
                Declare curr_byt as unsigned 16 bit integer = (bits_written) / 8
                Declare curr_bit as unsigned 8bit integer = (bits_written) % 8
```

```
        Declare the_bitval as unsigned 16bit integer = code & (00000001 << i)
        the_bitval = the_bitval >> i

        Begin if (the_bitval == 1)
                Declare writebyte as unsigned 16 bit integer = writebuf[curr_byt]
                Declare bit as unsigned 16 bit integer = writebyte | (00000001 << (curr_bit))
                writebuf[curr_byt] |= bit
                ++bits_written
        End if
        Begin else if (the_bitval == 0)
                Declare writebyte as unsigned integer 16 bit = writebuf[curr_byt]
                Declare unsigned integer 16 bit, bit = writebyte & (00000001 << (curr_bit))
                writebuf[curr_byt] |= bit
                ++bits_written
        End else if
End for

Declare i as integer  = 0
Begin for (i = 0, i < 8, i++)
        Begin if (bits_written == (offset)*8)
                write_bytes(outfile, writebuf, sizeof(writebuf))
                bits_written = 0
        End if
         Declare curr_byt as unsigned integer 8bit = (bits_written) / 8
        Declare curr_bit as unsigned integer 8bit = (bits_written) % 8
        Declare the_bitval as unsigned integer 8bit= sym & (00000001 << ((curr_bit - curr_bit) + i))
        the_bitval = the_bitval >> ((curr_bit - curr_bit) + i)
        Begin if (the_bitval == 1)
                Declare as unsigned integer 8bits writebyte = writebuf[curr_byt]
                Declare as unsigned integer 8bit = writebyte | (00000001 << curr_bit)
                writebuf[curr_byt] |= bit
                ++bits_written
        End if
        Begin else if (the_bitval == 0)
                unsigned integer 8bit writebyte = writebuf[curr_byt]
                unsigned integer 8bit, bit = writebyte & (00000001 << curr_bit)
                writebuf[curr_byt] |= bit
                 ++bits_written
        End else if
```

```
        End for
End buffer_pair


Begin flush_pairs(in outfile as integer)
        Begin if (bits_written != offset * 8)
            write_bytes(outfile, writebuf, bits_written)
        End if
End flush_pairs


Begin read_pair(in infile as integer, in code as pointer to unsigned 16 bit integer, in unsigned
unsigned 8bit integer sym, in bitlen as unsigned 8bit integer) {

        Declare boolean moreread = 1
        *code = 0
        *sym = 0

        Begin for (int i = 0, i < bitlen, i++)
                Begin if (bits_read == offset * 8)
                        read_bytes(infile, readbuf, sizeof(readbuf))
                        bits_read = 0
                End if
         Declare curr_byt as unsigned integer 8bit= readbuf[bits_read / 8]
         Declare curr_bit as unsigned integer 16bit = bits_read % 8
        Declare the_bitval as unsigned integer 16bit = curr_byt & (00000001 << i)

         the_bitval = the_bitval >> i
        Begin if (the_bitval == 1)
                Declare bit as unsigned integer 16bit = *code | (00000001 << curr_bit)
                *code |= bit
                ++bits_read
        End if
        Begin else if (the_bitval == 0)
                Bit as unsigned integer 16bit = *code & (00000001 << curr_bit)
                *code |= bit
                ++bits_read
        End else if
   End for
    Begin if (*code == STOP_CODE)
```

```
   moreread = 0
   return moreread
End if

  Begin for (integer i = 0, i < 8, i++)
        Begin if (bits_read == offset * 8)
              read_bytes(infile, readbuf, sizeof(readbuf))
              bits_read = 0
        End if

   Declare curr_byt as unsigned integer 16 bit = readbuf[bits_read / 8]
   Declare curr_bit as unsigned integer 8 bit = bits_read % 8
   Declare the_bitval as unsigned integer 8 bit = curr_byt & (00000001 << i)
   the_bitval = the_bitval >> i

   Begin if (the_bitval == 1)
        Declare bit as unsigned integer 8 bit = *sym | (00000001 << curr_bit)
        *sym |= bit
        ++bits_read
End if
 Begin else if (the_bitval == 0)
        Declare bit as unsigned integer 8 bit = *sym & (00000001 << curr_bit)
        *sym |= bit
        ++bits_read
    End else if
  End for
 return moreread
End read_pair

Begin buffer_word(in outfile as integer, in w as pointer to Word)
        Begin if (w != NULL)
              Begin if (bits_read != offset * 8)
                    write_bytes(outfile, readbuf, offset)
              End if
        End if
End buffer_word

Begin flush_words(in outfile as integer)
        Begin if (bits_read != 4096)
              write_bytes(outfile, readbuf, sizeof(readbuf))
        End if
```

End flush_words