CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Text I/O
Text I/O

- Within `<stdio.h>`:
  - Formatted text: `scanf()`/`printf()`
  - Characters: `getchar()`/`putchar()`
  - Strings/Lines: `fgets()`/`puts()`

  • NEVER EVER EVER USE `gets()`
Text I/O
fgets()

Syntax
```c
char *fgets(char *str, int count, FILE *stream);
```

- **str** is where received data is stored
  - Needs to be an array
- **count** is how many characters to process
  - Stops when \n or (count-1) chars are received
- **stream** is stdin
#include <stdio.h>

int main(void)
{
    // Create enough memory for a 50 char string
    char inputData[50 + 1];

    fgets(inputData, sizeof(inputData), stdin);
}
String Processing
String Processing

• Within `<string.h>`:
  – Examination
    • Length: `strlen()`
    • Comparing: `strcmp()`/`strncmp()`
    • Splitting: `strtok()`
  – Manipulation
    • Copying: `strncpy()` (Don't use `strcpy()`!)
    • Appending: `strncat()`
String Processing
strlen()

Syntax

```c
size_t strlen(const char *str);
```

- `str` is the string to calculate the length of
- `size_t` can be treated as an `int`

Examples

```c
int x = strlen("My string"); // x = 9
char str[] = "asdf";
int y = strlen(str); // y = 4
```
String Processing

`strcmp()`

**Syntax**

```c
int strcmp(const char *s1, const char *s2);
```

- Ignores size of the strings, purely alphabetical comparison.
- Return value is >0 if `s1` alphabetically before `s2`, 0 if they're equal, <0 if `s2` alphabetically before `s1`.

**Examples**

```c
char *s1 = "apple", *s2 = "zed";
int cmpResult = strcmp(s1, s2);
if (cmpResult > 0) {
    printf("apple > zed\n");
} else if (cmpResult == 0) {
    printf("apple == zed\n");
} else {
    printf("apple < zed\n");
}
```
String Processing

`strtok()`

Syntax

```
char *strtok(char *s1, const char *s2);
```

- `s1` (input/output), string to be tokenized
  - Will be modified!
- `s2` (input) – Delimiters

Examples

```c
char s1[] = "This is an example!";

char *firstToken = strtok(s1, " "); // firstToken = "This"

char *secondToken = strtok(NULL, " "); // secondToken = "is"

char *thirdToken = strtok(NULL, " "); // thirdToken = "an"

char *fourthToken = strtok(NULL, " "); // fourthToken = "example!"
```
String Processing

strtok() Details

Example

```c
char s1[] = "This is an example!";
```

`s1`\

<table>
<thead>
<tr>
<th>This</th>
<th>is</th>
<th>an</th>
<th>example!</th>
<th>\0</th>
</tr>
</thead>
</table>

Maxwell James Dunne
String Processing

strtok() Details

Example

```c
char s1[] = "This is an example!";
char *firstToken = strtok(s1, " ");
```

```
firstToken
s1

This \0 is an example ! \0
```

Maxwell James Dunne
String Processing

strtok() Details

Example

```c
char s1[] = "This is an example!";
char *firstToken = strtok(s1, " ");
char *secondToken = strtok(NULL, " ");
```

firstToken
s1

```
This is \0 is \0 an example ! \0
```

secondToken
String Processing

strtok() Details

Example

```c
char s1[] = "This is an example!";
char *firstToken = strtok(s1, " ");
char *secondToken = strtok(NULL, " ");
char *thirdToken = strtok(NULL, " ");
```

```
<table>
<thead>
<tr>
<th>firstToken</th>
<th>s1</th>
<th>thirdToken</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;This</td>
<td>is an</td>
<td>!</td>
</tr>
<tr>
<td></td>
<td>example!&quot;</td>
<td></td>
</tr>
</tbody>
</table>
```

secondToken
String Processing

`strtok()` Details

Example

```c
char s1[] = "This is an example!";
char *firstToken = strtok(s1, " ");
char *secondToken = strtok(NULL, " ");
char *thirdToken = strtok(NULL, " ");
char *fourthToken = strtok(NULL, " ");
```

```
firstToken
s1
```

```
This \0 is \0 an \0 example \0!
```

```
secondToken
thirdToken
fourthToken
```
String Processing

strncpy()

Syntax

```
char *strncpy(char *s1, const char *s2, size_t n);
```

- **s1** (output) – where the string will be copied to
- **s2** (input) - the string that to be copied
- **n** - how many characters can be copied
- Undefined if s1 and s2 overlap!

Examples

```
char s1[50];
strncpy(s1, "asdf", 4);          // s1 = "asdf\0"
strncpy(s1 + strlen(s1), "asdf", 4); // s1 = "asdfasdf\0"
```
String Processing

strncat()

Syntax

```
char *strncat(char *s1, const char *s2, size_t n);
```

- **s1** (input/output) - is the base string
- **s2** (input) - the string that will be appended
- **n** - how many characters can be appended
- Undefined if s1 and s2 overlap!

Examples

```
char s1[50] = "This is an example!";
strncat(s1, "asdf", 4);
```
\[ \begin{array}{ccc} 0 & 1 & 2 \\ \uparrow & \uparrow & \uparrow \\ a & b & c \end{array} \]
String Processing

- Within `<stdlib.h>`:
  - Conversion
    - Integer: `atoi()`, `xtoi()`
    - Floats: `atof()`
- Within `<stdio.h>`:
  - Conversion
    - Any: `sscanf()`
String Processing
atof()

Syntax

```c
double atof(const char *s);
```

- `s` (input) – The string to parse
- Returns the converted value or 0.0

Examples

```c
char s1[] = "1.03";
double x = atof(s); // y = 1.03

char s2[] = "efg";
double y = atof(s); // y = 0.0
```
Dynamic Memory

malloc()
free()
Dynamic Memory

Valgrind
Memory leaks

- If pointers returned by `malloc()` are lost, that memory is then "lost"
- Easy to do because this may not crash your program, possibly only causing errors over long periods of time

Example

```c
void MyFunc(void)
{
    Complex *x = malloc(sizeof(Complex));
    ...
}
```
Dynamic Memory
Memory leaks

• So for every pointer obtained from `malloc()`, there should be an equivalent `free()` at some point

Example

```c
void MyFunc(void)
{
    Complex *x = malloc(sizeof(Complex));
    ...
    free(x);
}
```
Dynamic Memory

When to use the Heap

• For unknown amounts of data
  – Arrays are always fixed-length at compile time

• When data needs to be accessible outside of the scope it was created in
  – Pointers need to be passed around
Pointers

Pointers to pointers
Pointers

Pointers to pointers

- Since pointers can point to any valid datatype, they can also point to other pointers

- No limit on levels of indirection
Pointers

Pointers to pointers

Example

```c
{  
    int x = 6;
    int *y = &x;
    int **z = &y;
    printf("%d\n", **z);
}
```

32-bit Data Memory

(RAM)

<table>
<thead>
<tr>
<th>Address</th>
<th>0000 0000 0x3F50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000 0006</td>
</tr>
<tr>
<td>X</td>
<td>0000 3F54</td>
</tr>
<tr>
<td>Y</td>
<td>0000 3F58</td>
</tr>
<tr>
<td>Z</td>
<td>0000 0000</td>
</tr>
<tr>
<td></td>
<td>0000 0000</td>
</tr>
<tr>
<td></td>
<td>0000 0000</td>
</tr>
<tr>
<td></td>
<td>0000 0000</td>
</tr>
<tr>
<td></td>
<td>0x3F60</td>
</tr>
<tr>
<td></td>
<td>0x3F64</td>
</tr>
<tr>
<td></td>
<td>0x3F68</td>
</tr>
<tr>
<td></td>
<td>0x3F6C</td>
</tr>
</tbody>
</table>

Output

6
Pointers

Passing by reference, again

- Passing by reference only allows persistently changing the value 1 level of indirection from the pointer and further
  - If a pointer is passed to a function, the data it points to can be altered
  - If a pointer-to-a-pointer is passed, the pointer it points to and the data that pointer points to can be altered
Pointers
Passing by reference, again

Example interrupt

```c
void MyFunc(int *x)
{
    *x = 6;
}

int main(void)
{
    int myInt;
    int *myIntPtr = &myInt;
    MyFunc(&myIntPtr);
}
```
Pointers
Passing by reference, again

Example interrupt

```c
void MyFunc(int **x)
{
    *x = malloc(sizeof(int));
    if (*x) {
        **x = 6;
    }
}

int main(void)
{
    int *myInt;
    MyFunc(&myInt);
}
```
Pointers swap \((**x, **y)\)

XOR swap

\[
\begin{align*}
x &= x \oplus y \\
y &= y \oplus x \\
x &= x \oplus y
\end{align*}
\]
Enums
Enumerations are integer data types that you can create with a limited range of values. Each value is represented by a symbolic constant that may be used in conjunction with variables of the same enumerated type.

- Enumerations:
  - Are unique integer data types
  - May only contain a specified list of values
  - Values are specified as symbolic constants
Enumerations

How to Create an Enumeration Type

• Creates an ordered list of constants
• If unspecified, each label’s value is one greater than the previous label

Syntax

```
enum typeName {label₀, label₁, ..., labelₙ}
```

Where compiler sets $label₀ = 0$, $label₁ = 1$, $labelₙ = n$

Example

```
enum weekday {SUN, MON, TUE, WED, THR, FRI, SAT};
```

Label Values:

- SUN = 0, MON = 1, TUE = 2, WED = 3, THR = 4, FRI = 5, SAT = 6
Enumerations

How to Create an Enumeration Type

- Any label may be assigned a specific value
- The following labels will increment from that value

**Syntax**

```c
enum typeName {label_0 = const_0, ..., label_n}
```

Where compiler sets `label_0 = const_0`, `label_1 = (const_0 + 1)`, ...

**Example**

```c
enum people {Rob, Steve, Paul = 7, Bill, Gary};
```

Label Values:

- Rob = 0
- Steve = 1
- Paul = 7
- Bill = 8
- Gary = 9
Enumerations
How to Create an Enumeration Type

• Any label may be assigned a specific value
• The following labels will increment from that value

Syntax

```plaintext
enum typeName {label_0 = const_0, ..., label_n}
```

Where compiler sets \( label_0 = const_0, label_1 = (const_0 + 1), \ldots \)

Example

```plaintext
enum people {Rob = 'a', Steve, Paul, Bill, Gary};
```

Label Values:

\( Rob = 'a', Steve = 'b', Paul = 'c', Bill = 'd', Gary = 'e' \)
Enumerations
How to Create an Enumeration Type

• Any label may be assigned a specific value
• The following labels will increment from that value

**Syntax**

```c
enum typeName {label_0 = const_0, ..., label_n}
```

Where compiler sets `label_0 = const_0, label_1 = (const_0 + 1), ...`

**Example**

```c
enum people {Rob = -4, Steve, Paul, Bill, Gary};
```

Label Values:

Rob = -4, Steve = -3, Paul = -2, Bill = -1, Gary = 0
Enumerations
How to Declare an Enumeration Type Variable

- Declared along with type:

```
enum typeName {const-list} varname_1,...;
```

- Declared independently:

```
enum typeName varName_1,...,varName_n;
```

Example
```
enum weekday {SUN, MON, TUE, WED, THR, FRI, FRI, SAT} today;
enum weekday day;  // day is a variable of type weekday
```
Enumerations

How to Declare a ‘Tagless’ Enumeration Variable

- No type name specified:

  Syntax

  ```
  enum {const-list} varName_1,...,varName_n;
  ```

- Only variables specified as part of the `enum` declaration may be of that type
- No type name is available to declare additional variables of the `enum` type later in code

Example

```
enum {SUN, MON, TUE, WED, THR, FRI, SAT} Today;
```
Enumerations
How to Declare an Enumeration Type with \texttt{typedef}

- Variables may be declared as type \texttt{typeName} without needing the \texttt{enum} keyword

\textbf{Syntax}

\begin{verbatim}
typedef enum \{const-list\} typeName;
\end{verbatim}

- The enumeration may now be used as an ordinary data type (compatible with \texttt{int})

\textbf{Example}

\begin{verbatim}
typedef enum \{SUN, MON, TUE, WED, THR, FRI, SAT\} Weekday;

Weekday day; // Variable of type weekday
\end{verbatim}
Enumerations

How to Use an Enumeration Type Variable

If enumeration and variable have already been defined:

Syntax

\[ \text{varName} = \text{label}_n; \]

- The labels may be used as any other symbolic constant
- Variables defined as enumeration types must be used in conjunction with the type’s labels or equivalent integer

Example

```c
enum weekday {SUN, MON, TUE, WED, THR, FRI, SAT};
enum weekday day;

day = WED;
day = 6; // May only use values from 0 to 6
if (day == WED) {
...}
```
Enumerations

Proper formatting

Example

typedef enum {
    SUN,  // foo
    MON,  // 600
    TUE,
    WED,
    THR,
    FRI,
    SAT
} Weekday;

Weekday day = WED;
Enumerations
Proper formatting

Example

typedef enum {  
    SUN,  
    MON,  
    TUE,  
    WED,  
    THR,  
    FRI,  
    SAT  
} Weekday;

Weekday day = 3; // No compilation warning/error
Enumerations
Datatype usage

Example

typedef enum {
    SUN,
    MON,
    TUE,
    WED,
    THR,
    FRI,
    SAT
} Weekday;

void PrintDayName(Weekday d)
{
    if (d == SUN) {
        printf("Sun\n");
    } ... 
}

PrintDayName(WED); // No compilation warning/error
enum
{
    X = base
    Y = 6 * ase + 2
    Z
}

#define X
#define Y

...
Enumerations

Why enumerations?

- Enumerations are a proper datatype as well as the possible values for them
- Some compile-time checking
- Doesn't do text replacement, done during the compiler stage
- Use for a group of related values
Interrupts
Interrupts

- High-priority alerts that an event requires immediate attention
- Generally interrupts can be assigned priorities
- Event is handled by an Interrupt Service Routine (ISR)
Interrupts

- ISR is a special function that is written by the developer, but called directly by the processor.
- ISRs have no inputs or outputs
  - All processing through global/module-level variables
- ISRs are written a specific way and the processor is told they have been implemented by the compiler/developer.
Interrupts

Traps

- Software interrupts are generally referred to as exceptions or traps
- Examples:
  - Division by zero
  - Invalid address dereference
  - Debugging breakpoint
  - Stack overflow
Example interrupt

```c
void _ISR IsrName(void)
{
    // Process data from the interrupt

    // Store results in global/module variable

    // Clear interrupt flag
}
```
void _ISR Uart1TxInterrupt(void)
{
    // Stall until transmission finishes
    while (!U1STAbits.TRMT);

    // Continue transmitting next batch of data
    Uart1StartTransmission();

    // Clear interrupt flag
    IFS0bits.U1TXIF = 0;
}
Interrupts

Calling

Example program

```c
int main(void)
{
    int x = 20;
    int y;
    y = x / 2;
}
```

Interrupt: UART1 Post-transmission

```c
void _ISR __U1TXInt(void)
{
    IFS0bits.U1TXIF = 0;
}
```
Example program:

```c
int main(void)
{
    int x = 20;
    int y;
    _U1TXInt();
    y = x / 2;
}
```

Interrupt: UART1 Post-transmission:

```c
void _ISR_U1TXInt(void)
{
    IFS0bits.U1TXIF = 0;
}
```
Interrupts

- Interrupts are important events that happen in real-time
- ISRs are the functions that handle these events
- ISRs are called outside of regular program execution order