CIS 314 Lab

Introduction to C

Brian Gravelle & Shreeya Khadka
Programming Process

- Write Program
- Compile Program
- Run Program
- Debug Program
Basic C variable types

• There are five basic data types in C
  ‣ Char: ‘a’
    • A single byte capable of holding one character in the local character set
  ‣ Int: 3
    • An integer of unspecified size
  ‣ Float: 3.14
    • Single-precision floating point
  ‣ Double: 3.1415926
    • Double-precision floating point
  ‣ Void: Valueless special purpose type
# Basic C variable types

<table>
<thead>
<tr>
<th>Type (32 bit)</th>
<th>Smallest Value</th>
<th>Largest Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>short int</td>
<td>-32,768(-2^{15})</td>
<td>32,767(2^{15}-1)</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>0</td>
<td>65,535(2^{16}-1)</td>
</tr>
<tr>
<td>int</td>
<td>-2,147,483,648(-2^{31})</td>
<td>2,147,483,648(2^{31}-1)</td>
</tr>
<tr>
<td>unsigned int</td>
<td>0</td>
<td>4,294,967,295</td>
</tr>
<tr>
<td>long int</td>
<td>-2,147,483,648(-2^{31})</td>
<td>2,147,483,648(2^{31}-1)</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>0</td>
<td>4,294,967,295</td>
</tr>
</tbody>
</table>
Variable assignment

• In C variables must be declared
• They are given values through assignments
• Assignment is done with the '==' operator

Declarations

```c
int number_of_students;
float average_gpa;
```

Assignments

```c
number_of_students = 12;
average_gpa = 3.9;
```
Variable assignment

- In C variables must be declared
- They are given values through assignments
- Assignment is done with the '==' operator

Declarations

```c
int number_of_students,
float average_gpa;
```

Assignments

```c
number_of_students = 12;
average_gpa = 3.9;
```
#include <stdio.h> /* Header files */

int main(void) {
    printf ("Hello World!\n") ;
    return 0;
}
C Program compilation

Compile: gcc – o myhello hello.c

Run: ./myhello
C Program Analysis

• `#include <stdio.h> /* Header files */`
  ‣ It is a preprocessor directive
  ‣ It tells computer to load contents of the file
  ‣ It allows standard input/output operations

• Comments are used to describe program
  ‣ Text surrounded by /* and */ is ignored by computer
  ‣ Lines starting with // are also ignored
C Program Analysis

• int main (void)
  ‣ C programs contain one or more functions, exactly one of which must be main
  ‣ Parenthesis used to indicate a function
  ‣ int means that main "returns" an integer value

• Braces ({ and }) indicate a block
  ‣ Bodies of all functions must be contained in braces

• printf ("Hello World!\n")
  ‣ printf and scanf functions
C Program Analysis

• `printf`
  ▸ Sends output to standard out
  ▸ General form
    • `printf(format descriptor, var1, var2, ...);`
  ▸ `printf("%s\n", "Hello world");`
    • Translation: Print hello world as a string followed by a newline character
  ▸ `printf("%d\t%f\n", j, k);`
    • Translation: Print the value of the variable j as an integer followed by a tab followed by the value of floating point variable k followed by a new line
C Program Analysis

- **scanf**
  - Gets inputs from user
  - **General form**
    - `scanf(format descriptor, &var1, &var2, ...);
  - `scanf("%f", &i);`
    - Translation: Get floating point input i from user
  - `scanf("%d %f\n", &j, &k);`
    - Translation: Get the value of the variable j as an integer followed by the value of floating point variable k from user
    - Blocks program until user enters input
C Program Analysis

• Some special characters are not visible directly in the output stream

• These begin with an escape character (\):
  ‣ \n    newline
  ‣ \t    horizontal tab
  ‣ \a    alert bell
  ‣ \v    vertical tab
C Program Operations

- Arithmetic operators
  - `+` “plus”
  - `-` “minus”
  - `*` “times”
  - `/` “divided by”

```c
#include <stdio.h> /* Header files */

int number1, number2, number3;

int main(void) {
    scanf("Enter number1: %d", &number1);
    scanf("Enter number2: %d", &number2);
    number3 = number1 + number2;
    printf("Number1 + number2 = %d\n", number3);

    number3 = number1 - number2;
    printf("Number1 - number2 = %d\n", number3);

    number3 = number1 * number2;
    printf("Number1 * number2 = %d\n", number3);

    number3 = number1 / number2;
    printf("Number1 / number2 = %d\n", number3);
    return 0;
}
```
C Program Comparators

- **Relational operators:**
  - `==` “is equal to”
  - `!=` “is not equal to”
  - `>` “greater than”
  - `<` “less than”
  - `>=` “greater than or equal to”
  - `<=` “less than or equal to”
C Program Logical Operators

- There are two logical operators in C

- `||` “logical or”
  - An expression formed with `||` evaluates to true if any one of its components is true

- `&&` “logical and”
  - An expression formed with `&&` evaluates to true if all of its components are true
Advance Data types

- In C
  - Arrays (a list of data (all of the same data type!))
    - int grades[] = {94, 78, 88, 90, 93, 87, 59};
  - Structures (a collection of named data referring to a single entity)

```c
struct Student {
    char Name [50] ;
    int id;
    float GPA;
    char major [25] ;
};
```
Advance Data types

• Pointers in C
  ‣ Pointers are memory addresses
  ‣ Every variable has a memory address
  ‣ Symbol & means “take the address of” e.g., &x
  ‣ Symbol * means “take the value of” e.g., *p
  ‣ Symbol * is also used to denote a pointer type e.g., int *q;
Advance Data types

• Pointers in C
  ‣ Declaration of integer pointers and an integer number
    • int * pointer1, pointer2;
    • int number1;
  ‣ Setting pointer1 equal to the address of number1
    • pointer1 = &number1;
  ‣ Setting pointer2 equal to pointer1
    • pointer2 = pointer1;
Conditional execution

- Conditional constructs provide the ability to control whether a statement list is executed.

- If statement
  - if (Expression)
  - Action

Expression

true

false

Action
Conditional execution

- Conditional constructs provide the ability to control whether a statement list is executed

- If statement
  - If
  - if (Expression)
    - Action

int num1;
int num2;
printf("Enter two integers: ");
scanf("%d %d", num1, num2);
if (num1 > num2) {
  int remember_num1 = num1;
  num1 = num2;
  num2 = remember_num1;
}
printf("Inputs in sorted order: %d \t %d", num1, num2 );
Conditional execution

- Conditional constructs provide the ability to control whether a statement list is executed

- If statement
  - if-else
    - if (Expression)
      - Action1
    - else
      - Action2
Conditional execution

- Conditional constructs provide the ability to control whether a statement list is executed

- **If statement**
  - if-else
    - if (Expression)
      - Action1
    - else
      - Action2

```c
int num1;
int num2;
printf("Enter two integers: ");
scanf("%d %d", num1, num2);

int max;
if (num1 > num2) {
    max = num1;
} else {
    Max = num2
}
printf("Maximum number is: %d", max);
```
Conditional execution

• For loop
• Do loop
• While loop
• Do while loop
Iteration

• For loops

  ▶ The general format when using for loops is

    • for ( initialization; LoopContinuationTest; increment )

    Statement

    Initialize variable

    Condition
    Test the variable

    true
    false

    statement

    Increment variable
Iteration

• For loops
  ‣ The general format when using for loops is
    • for ( initialization; LoopContinuationTest; increment )
      Statement

int counter;  
for(counter = 1; counter <= 10; counter++)
  printf("Current counter value \%d\n", counter);
• While loop
  ‣ while loop repeated until condition becomes false

• Initialization;

while ( loopContinuationTest ){

  statement

  increment;

}
Iteration

- **While loop**
  
  - while loop repeated until condition becomes false

- Initialization;
  
  ```
  int counter = 1; // initialization
  while (counter <= 10){ // repetition
    // condition
    printf("Current counter value \%d \n", counter);
    counter++; // increment
  }
  ```

- **Diagram**

  ![Diagram of While Loop](image)
Iteration

• Do while
  ‣ do/while repetition structure is similar to the while structure
  ‣ Condition for repetition tested after the body of the loop is executed

• do {
  statement
} while ( condition );
Recursion

• Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first

• Recursion is a technique that solves a problem by solving a smaller problem of the same type
Recursion

- Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first
- Recursion is a technique that solves a problem by solving a smaller problem of the same type
- The technique ends up with functions that call themselves (recursive functions)
Recursion

- Factorial function
  - Iterative implementation

```c
int Factorial(int n) {
    int count;
    int fact = 1;
    for(count = 2; count <= n; count++)
        fact = fact * count;
    return fact;
}
```
Recursion

• Factorial function
  ‣ Recursive implementation

```c
int Factorial(int n) {
    if (n==0)  // base case
        return 1;
    else
        return n * Factorial(n-1);
}
```
Next Lab Module

• MIPS & SPIM