C Programming Basics

Ritu Arora

Texas Advanced Computing Center

February 20th, 2013

Email: rauta@tacc.utexas.edu

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Creating a C Program

- Have an idea about what to program
- Write the source code using an editor or an Integrated Development Environment (IDE)
- Compile the source code and link the program using a C compiler
- Fix errors, if any
- Run the program and test it
- Fix bugs, if any

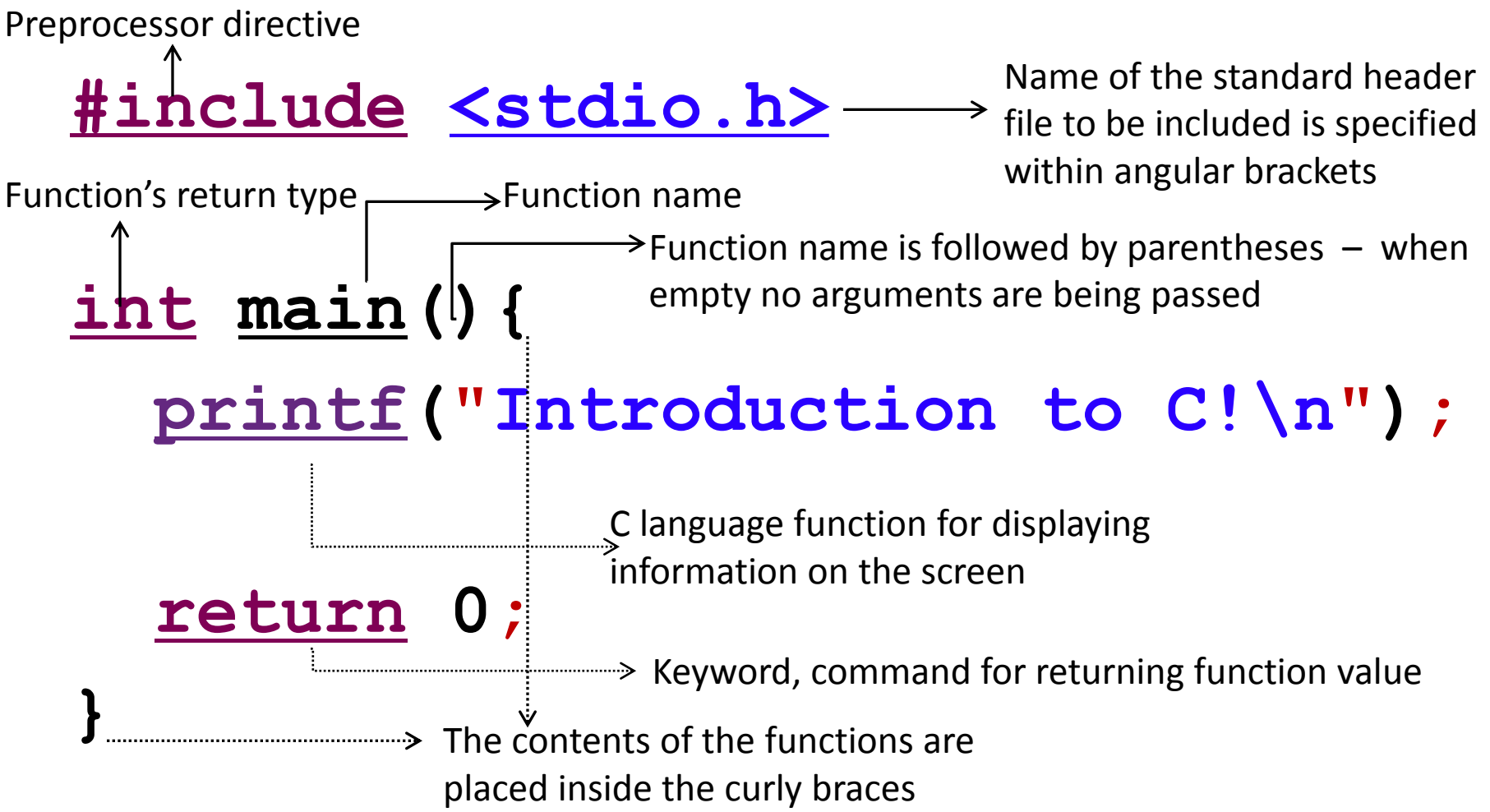
Write the Source Code: firstCode.c

```
#include <stdio.h>

int main(){

   printf("Introduction to C!\n");

   return 0;

}
```

Output:

```
Introduction to C!
```

Understanding firstCode.c



Text strings are specified within "" and every statement is terminated by ;
Newline character is specified by \n

Save-Compile-Link-Run

- Save your program (source code) in a file having a “c” extension.

  Example, `firstCode.c`

- Compile and Link your code (by default, GCC automatically does the linking)

```
gcc -o firstCode firstCode.c
```

- Run the program

```
./firstCode
```

Repeat the steps above every time you fix an error!

Different Compilers

- Different commands for different compilers (*e.g.*, **`icc`** for intel compiler and **`pgcc`** for pgi compiler)
  - GNU C program

```
gcc -o firstCode firstCode.c
```

  - Intel C program

```
icc -o firstCode firstCode.c
```

  - PGI C program

```
pgcc -o firstCode firstCode.c
```

- To see a list of compiler options, their syntax, and a terse explanation, execute the compiler command with the -help or --help option

Summary of C Language Components

- Keywords and rules to use the keywords
- Standard header files containing functions like **`printf`**
- Preprocessor directives for including the (standard) header files
- Function **`main`**
- Parentheses and braces for grouping together statements and parts of programs
- Punctuation like `;`
- Operators like `+`
- All the above and more to come make up the syntax of C

Pop Quiz
(add the missing components)

```
_________ <stdio.h>

int main()__

  printf("Introduction to C!\n") __

  printf("This is a great class!\n");

  return 0;


__
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Warnings, Errors and Bugs

- Compile-time warnings
  - Diagnostic messages
- Compile-time errors
  - Typographical errors: pirntf , $include
- Link-time errors
  - Missing modules or library files
- Run-time errors
  - Null pointer assignment
- Bugs
  - Unintentional functionality

Find the Error: error.c

```
#include <stdio.h>
int main(){
   printf("Find the error!\n")
   retrun(0);
}
```

Error Message (compile-time error)

```
**** Internal Builder is used for build****
gcc -O0 -g3 -Wall -c -fmessage-length=0 -oerror.o
..\error.c
..\error.c: In function 'main':
..\error.c:4:3: error: expected ';' before 'retrun'
..\error.c:5:1: warning: control reaches end of non-
void function
Build error occurred, build is stopped
Time consumed: 148  ms.
```

Find the Error: error.c

```
#include <stdio.h>
int main(){
   printf("Find the error!\n");
   retrun 0;
}
```

Error Message (link-time error)

```
gcc -o error error.c
…
..\error.c:4:3: warning: implicit declaration of function 'retrun'
…
gcc -oCTraining.exe error.o
error.o: In function `main':
C:\Users\ra25572\workspace\CTraining\Debug/../error.c:4: undefined reference to `retrun'
collect2: ld returned 1 exit status
Build error occurred, build is stopped
Time consumed: 436  ms.
```

Find the Error: error2.c

```
#include < stdio.h >
int main(){
   printf("Find the error!\n");
   return 0;
}
```

Error Message (compile-time error)

```
gcc -o error2 error2.c
..\error2.c:1:21: fatal error:  stdio.h : No such file or directory
compilation terminated.
Build error occurred, build is stopped
Time consumed: 98  ms.
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Comments and New Line: rules.c

```
/*
 * rules.c
 * this is a multi-line comment
 */
#include <stdio.h>
int main(){
  printf("Braces come in pairs.");
  printf("Comment tokens come in pairs.");
  printf("All statements end with semicolon.");
  printf("Every program has a main function.");
  printf("C is done mostly in lower-case.");
  return 0;
}
```

Output of rules.c

```
Braces come in pairs.Comment tokens come in
pairs.All statements end with a
semicolon.Every program must have a main
function.C is done mostly in lower-case.
```

Output looks odd! We want to see a new line of text for every `printf` statement.

Comments and New Line: rules.c

```
/*
 * rules.c
 * this is a multi-line comment
*/
#include <stdio.h>
int main(){
 /* notice the \n in the print statements */
 printf("Braces come in pairs.\n");
 printf("Comment tokens come in pairs.\n");
 printf("All statements end with semicolon.\n");
 printf("Every program has a main function.\n");
 printf("C is done mostly in lower-case.\n");
 return 0;
}
// this is another way to specify single-line comments
```

Output of rules.c

```
Braces come in pairs.
Comment tokens come in pairs.
All statements end with a semicolon.
Every program must have a main function.
C is done mostly in lower-case.
```

The output looks better now!

Do-It-Yourself Activity

- Learn the various ways in which you can print and format values of various data types.

- For example:
  - How would you print an integer?
  - How would you print a value of type double with precision of 8 places after the decimal?

- Reference:
  - http://www.cplusplus.com/reference/clibrary/cstdio/printf/

Some C Language Keywords

Category	Keywords
Storage class specifiers	`auto register static extern typedef`
Structure & union specifiers	`struct union`
Enumerations	`enum`
Type-Specifiers	`char double float int long short signed unsigned void`
Type-Qualifiers	`const volatile`
Control structures	`if else do while for break continue switch case default return goto`
Operator	`sizeof`
Deprecated keywords	`fortran entry`
Other reserved words	`asm bool friend inline`

Variables

- Information-storage places
- Compiler makes room for them in the computer's memory
- Can contain string, characters, numbers *etc.*
- Their values can change during program execution
- All variables must be declared before they are used and must have a data type associated with them
- Variable must be initialized before they are used

Data Types

- Data types specify the type of data that a variable holds

- Categories of data types are:
  - **Built-in:** `char double float void int (short long signed unsigned)`
  - **User-defined:** `struct union enum`
  - **Derived:** `array function pointer`

- We have already seen an example code in which an integer data type was used to return a value from a function:

```
int main()
```

- Compiler-dependent range of values associated with each type. For example: an `int` can have a value in the range
  - **–32768** to **32767** on a 16-bit computer or
  - **–2147483647** to **2147483647** on a 32-bit computer

Identifiers

- Each variable needs an identifier (or a name) that distinguishes it from other variables
- A valid identifier is a sequence of one or more letters, digits or underscore characters
  - Note: you cannot begin with a digit
- Keywords cannot be used as identifiers

Variable Declaration

- Declaration is a statement that defines a variable
- Variable declaration includes the specification of data type and an identifier. Example:

```
int number1;
float number2;
```

- Multiple variables can be declared in the same statement

```
int x, y, z;
```

- Some types of data can be signed or unsigned
- Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values

```
signed double temperature;
```

Variable Initialization

- A variable can be assigned a value when declared
  - Assignment operator is used for this purpose
  - `int x = 10;`

- More examples
  - `char x = 'a';`
  - `double x = 22507385850720l4.e23;`
  - `float x = 10.11;`

- `void` cannot be used to declare a regular variable
  - It is used as a return type of a function or as an argument of a function

Example of Updating Variables: myAge.c

```
#include <stdio.h>
int main(){
   int age;
   age = 10;
   printf("Initial value of age is: %d\n", age);
   age = 20;
   printf("Updated value of age is: %d\n", age);
   age = age + 20;
   printf("New updated value of age is: %d\n", age);
   return 0;
}
```

Output:

```
Initial value of age is: 10
Updated value of age is: 20
New updated value of age is: 40
```

Scope of Variables

- A variable can be either of global or local scope
  - Global variables are defined outside all functions and they can be accessed and used by all functions in a program file
  - A local variable can be accessed only by the function in which it is created
- A local variable can be further qualified as **`static`**, in which case, it remains in existence rather than coming and going each time a function is called
  - **`static int x = 0;`**
- A **`register`** type of variable is placed in the machine registers for faster access – compilers can ignore this advice
  - **`register int x;`**

Constants and Constant Expressions

- The value of a constant never changes
  - `const double e = 2.71828182;`

- Macros
  - `#define MAXRECORDS 100`
  - In the code, identifiers (`MAXRECORDS`) are replaced with the values (`100`)
  - Helps to avoid hard-coding of values at multiple places
  - Example: `char records[MAXRECORDS + 1];`
  - Can be used at any place where constants can be used

- Enumeration is a list of constant values
  - `enum boolean {NO , YES};`

Expressions containing constants are evaluated at compile-time

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Reading Keyboard Input: readInput1.c

```
#include <stdio.h>
int main(){
   char myName[50];
   printf("What is your name?");
   fflush(stdout);
   scanf("%s", &myName);
   printf("Hello %s!", &myName);
   return 0;
}
```

`scanf` function is used to read the keyboard input

`fflush` flushes the contents of the output buffer

Understanding readInput1.c

```
#include <stdio.h>
int main(){

   char myName[50];

   printf("What is your name?");

   fflush(stdout);

   scanf("%s", &myName);

   printf("Hello %s!", &myName);
   return 0;
}
```

This is a **variable declaration** for string type and **myName** is a string variable. It provides storage for the information you enter. Note the usage of **char**.

Explicit flushing of the output stream

Function to read the value from keyboard and store it in computer's memory

More Information on `scanf`

- Function to read information from the keyboard

```
scanf("%s", &myName);
```

- First parameter is a type-specifier
  - `%s` is a type-specifier that is used if input data is string or text.
  - other type-specifiers are %c for character, %d for decimal, %f for float, %o for octal, %x for hexadecimal

- The second parameter is the address of the variable that would store the value being input from the keyboard
  - **`myName`** is the string variable for storing the input value
  - Ampersand (&) before the variable name helps `scanf` find the location of the string variable in memory

More functions for I/O

- **`gets`** function is used to read the keyboard input (*i.e.,* standard input stream)

```
gets(myName);
```

Warning: keyboard overflow! Avoid using it.

- **`puts`** function is used to print text on the screen (*i.e.,* standard output stream)

```
puts(myName);
puts("Hello Ritu");
```

Unlike **`printf`**, it always displays a newline character and can print only one variable or a string

More functions for I/O

- `getchar()` function is used to read a single character from the keyboard
  - It causes the program to pause until a key is typed at the keyboard and Enter is pressed after that
  - More on this syntax later

- `putchar(c)` function displays the character on the screen
  - `c` can be a character constant in single quotes or a variable name

- More on variables later

String Variables

- Numeric values can be assigned by using the "=" sign but string values cannot be assigned using the "=" sign

```
char myName[50];
myName = "Ritu"; // this is wrong
```

- Three ways to assign values to strings

```
scanf("%s", &myName);
gets(myName);
strcpy(myName,"Ritu");
```

- Function `strcpy`
  - It is defined in the header file `string.h` and hence needs to be included
  - It copies the value of one string to another

strcpy Example: writeStringChar.c

```
#include <stdio.h>
#include <string.h>
int main(){
   char myName[50];
   char c;
   strcpy(myName,"Ritu");
   c = 'a';
   printf("Your name is: %s\n", myName);
   printf("The character is: %c \n", c);
   return 0;
}
```

Output:

```
Your name is: Ritu
The character is: a
```

Numbers Entered From Keyboard

- Keyboard input is read as a string
- The integer 25 is different from text “25” entered via keyboard
- Convert string to integer by using the **`atoi`** function
  - It is defined in the header file **`stdlib.h`**
  - The string to be converted by this function should begin with a number
- For other conversion functions see:

http://en.wikibooks.org/wiki/C_Programming/C_Reference/stdlib.h

String to Integer Conversion: strToInt.c

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int age;
 char enterAge[8];
 printf("How old is your friend?\n");
 fflush(stdout);
 gets(enterAge); // enter the value for age
 age = atoi(enterAge);
 printf("Your friend's age is: %d", age);
 return 0;
}
```

----------> Note: string to integer conversion

Output:

```
How old is your friend?
22
Your friend's age is: 22
```

Pop Quiz
(Reflect on this & ask questions, if any)

- How will you use **scanf** to read different data types?

- How will you instruct the compiler to ignore certain lines of code during program compilation?

- Is the following statement correct?

```
printf("%s, your color is: %s","red");
```

- Fill in the blanks(__):

```
scanf("%__", __myIntegerNumber);
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Operators

- Arithmetic: +, -, /, *, %, ++, --, =
- Relational: a == b, a != b, a > b, a < b, a >= b, a <= b
- Logical: !a, a && b, a || b
- Member and Pointer: a[], *a, &a, a->b, a.b
- Other: **`sizeof`**
- Bitwise: ~a, a&b, a|b, a^b, a<<b, a>>b
- More about operators and precedence:

http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

Parentheses and Precedence: checkParentheses.c

```c
#include <stdio.h>
int main(){
 int total;
 //multiplication has higher precedence than subtraction
 total=100 - 25*2;
 printf("The total is: $%d \n",total);
 //parentheses make a lot of difference!
 total=(100 - 25)*2;
 printf("The total is: $%d \n",total);
 return 0;
}
```

```
Output:
The total is: $50
The total is: $150
```

`sizeof` Operator Example: testSize.c

```
#include <stdio.h>
int main(){
 char c;
 int x;
 printf("Size of variable c is %d bytes\n", sizeof(c));
 printf("Size of variable x is %d bytes\n", sizeof(x));
 return 0;
}
```

Note: Byte sizes of variables can be found by using `sizeof` operator

```
Output:
Size of variable c is 1 bytes
Size of variable x is 4 bytes
```

Note: Declaring a character variable (`char c;`) is different from declaring a string (`char myName[50];`)

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Control Structures

- **Sequence Structure** is a sequence of statements
- **Selection Structure** used for branching
- **Loop Structure** used for iteration or repetition

Conditional Expressions

- Use **`if-else`** or ternary operator (**`?:`**)

```
if (a > b){
   z = a;
} else {
   z = b;
}
```

```
z = (a > b) ? a : b ; //z = max (a, b)
```

if-else: Logical Expressions

```
if(temp > 75 && temp < 80){
 printf("It's nice weather outside\n");
}
```

```
if (value == 'e' || value == 'n' ){
  printf("\nExiting the program.\n");
} else {
  printf("\nIn the program.\n");
}
```

Decision Making, Multi-Way Decisions

- Decisions are expressed by **`if-else`** where the **`else`** part is optional

```
if (expression)
   statement1
else
   statement2
```

- Multi-way decisions are expressed using **`else-if`** statements

```
if (expression1)
   statement1
else if (expression2)
   statement2
else
   statement3
```

Multi-Way Decision

- The **`switch`** statement is a multi-way decision
- It tests whether an expression matches one of a number of constant integer values, and branches accordingly

```
switch (expression){
  case const-expression1: statements1
  case const-expression2: statements2
  default: statements3
}
```

Multi-Way Decision Example 1: multiWay1.c

```
char c;
//other code
c = getchar();
if(c=='1')
    printf("Beverage\nThat will be $8.00\n");
else if(c=='2')
    printf("Candy\nThat will be $5.50\n");
else if(c=='3')
    printf("Hot dog\nThat will be $10.00\n");
else if(c=='4')
    printf("Popcorn\nThat will be $7.50\n");
else{
    printf("That is not a proper selection.\n");
    printf("I'll assume you're just not hungry.\n");
    printf("Can I help whoever's next?\n");
}
```

the character read from the keyboard is stored in variable c

If multiple statements depend upon a condition, use { }

//This is just a code snippet. For complete program, see file multiWay1.c

Output of multiWay1.c

```
Please make your treat selection:
1 - Beverage.
2 - Candy.
3 - Hot dog.
4 - Popcorn.
3 <enter>
Your choice:Hot dog
That will be $10.00
```

Multi-Way Decision Example 2: multiWay2.c

```
c = getchar();
switch(c){
  case '1':
    printf("Beverage\nThat will be $8.00\n");
    break;
  case '2':
    printf("Candy\nThat will be $5.50\n");
    break;
  case '3':
    printf("Hot dog\nThat will be $10.00\n");
    break;
  case '4':
    printf("Popcorn\nThat will be $7.50\n");
    break;
  default:
    printf("That is not a proper selection.\n");
    printf("I'll assume you're just not hungry.\n");
    printf("Can I help whoever's next?\n");
}
```

//This is just a code snippet. For complete program, see file multiWay2.c

Loops

- For repeating a sequence of steps/statements
- The statements in a loop are executed a specific number of times, or until a certain condition is met
- Three types of loops
  - `for`
  - `while`
  - `do-while`

`for` Loop

```
for (start_value; end_condition; stride)
    statement;

for (start_value; end_condition; stride) {
    statement1;
    statement2;
    statement3;
}
```

`for` Loop Example 1: forLoop.c

```
#include <stdio.h>
int main(){
  int i;
  for(i = 0 ; i <= 10 ; i = i+2){
    printf("What a wonderful class!\n");
  }
  return 0;
}
```

Output:

```
What a wonderful class!
What a wonderful class!
What a wonderful class!
What a wonderful class!
What a wonderful class!
What a wonderful class!
```


TACC
THE UNIVERSITY OF TEXAS AT AUSTIN

`for` Loop Example 2

```
#include <stdio.h>
int main(){
   int i, sum;
   sum = 0;
   for(i = 1 ; i <= 100 ; i = i+1){
      sum = sum + i;
   }
   printf("Sum of first 100 numbers is: %d ", sum);
   return 0;
}
```

Output:

```
Sum of first 100 numbers is: 5050
```

Did you notice how multiple variables can be declared in the same line?

`while` Loop

- The while loop can be used if you don't know how many times a loop should run

```
while (condition_is_true){
   statement (s);
}
```

- The statements in the loop are executed until the loop condition is true

- The condition that controls the loop can be modified inside the loop (this is true in the case of `for` loops too!)

`while` Loop Example: whileLoop.c

```
#include <stdio.h>
int main(){
  int counter, value;
  value = 5;
  counter = 0;
  while ( counter < value ){
    counter++;  <-- Equivalent to counter = counter +1;
    printf("counter value is: %d\n", counter);
  }
  return 0;
}
```

Output:

```
counter value is: 1
counter value is: 2
counter value is: 3
counter value is: 4
counter value is: 5
```


TACC
THE UNIVERSITY OF TEXAS AT AUSTIN

do-while Loop

- This loop is guaranteed to execute at least once

```
do{
 statement (s);
}
while(condition_is_true);
```

do-while Example: doWhile.c

```
#include <stdio.h>
int main(){
  int counter, value;
  value = 5;
  counter = 0;
  do {
    counter++;
    printf("counter value is: %d\n", counter);
  } while ( counter < value);
  return 0;
}
```

Note the semi-colon after specifying while

Output same as that of the while loop program shown earlier

Keyword: `break`

- **`break`** is the keyword used to stop the loop in which it is present

```
for(i = 10; i > 0; i = i-1){
    printf("%d\n",i);
    if (i < 5){
      break;
    }
}
```

Output:

```
10
9
8
7
6
5
4
```

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

`continue` Keyword: myContinue.c

- `continue` is used to skip the rest of the commands in the loop and start from the top again
- The loop variable must still be incremented though

```
#include <stdio.h>
int main(){
   int i;
   i = 0;
   while ( i < 20 ){
      i++;
      continue;
      printf("Nothing to see\n");
   }
   return 0;
}
```

The printf statement is skipped, therefore no output on screen.

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

C Language Functions

- Functions are self-contained blocks of statements that perform a specific task

- Written once and can be used multiple times
  - Promote code reuse
  - Make code maintenance easy

- Two steps involved
  - Write the function
    - Function definition
    - Function declaration or prototype
  - Invoke or call the function

- Two types of functions
  - Standard or library or built-in
  - User-Defined

Standard Functions

- These functions are provided to the user in library files
- In order to use the functions, the user should include the appropriate library files containing the function definition
- Example
  - `scanf`
  - `printf`
  - `gets`
  - `puts`
  - `strcpy`

User-Defined Functions: myFunction.c

```
#include <stdio.h>
---------------Defining the function add
void add(){
   int a, b, c;
   printf("\n Enter Any 2 Numbers : ");
   fflush(stdout);
   scanf("%d %d",&a,&b);
   c = a + b;
   printf("\n Addition is : %d",c);
}
int main(){
   add();  <-- Invoking the function add twice from
   add();  <--  function main
   return 0;
}
```

Function Prototype: myFctPrototype.c

```
#include <stdio.h>

void add();

int main(){
 add();
 return 0;
}

void add(){
int a, b, c;
printf("\n Enter Any 2 Numbers : ");
fflush(stdout);
scanf("%d %d",&a,&b);
c = a + b;
printf("\n Addition is : %d",c);
}
```

Function Prototype or Declaration: useful when the function is invoked before its definition is provided

Invoking the function add

Defining the function add that does not return a value – note void

Categories of Functions

- Functions that take no input, and return no output
- Functions that take input and use it but return no output
- Functions that take input and return output
- Functions that take no input but return output

Sending Input Values To Functions

- Determine the number of values to be sent to the function
- Determine the data type of the values that needs to be sent
- Declare variables having the determined data types as an argument to the function
- Use the values in the function
- Prototype the function if its definition is not going to be available before the place from where it is invoked
- Send the correct values when the function is invoked

Passing Values to Functions: passValue1.c

```
#include <stdio.h>
void add(int a, int b){<-- Formal Parameters: a, b
   int c;
   c = a + b;
   printf("\n Addition is : %d",c);
}
int main(){
   int a, b;
   printf("\n Enter Any 2 Numbers : ");
   fflush(stdout);
   scanf("%d %d",&a,&b);
   add(a, b); <-- Actual Parameters: a, b
   return 0;
}
```

Note: The variables used as formal and actual parameters can have different names.

Passing Values to Functions: passValue2.c

```
#include <stdio.h>
#include <stdlib.h>
void add(int a, int b){
 //same code as in the previous slide
}
int main(int argc, char *argv[]){
 int a, b;
 if ( argc != 3 ){
  printf("\nInsufficient num. of arguments.\n");
  printf( "\nUsage:%s <firstNum> <secondNum>", argv[0]);
 }else{
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    add(a, b);
 }
 return 0;
}
```

Code Snippet From passValue2.c

Notice that main has two arguments

```
int main(int argc, char *argv[]){
 int a, b;
 if ( argc != 3 ){
  printf("\nInsufficient num. of arguments.\n");
  printf( "\nUsage:%s <firstNum> <secondNum>", argv[0]);
 }else{
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    add(a, b);
 }
 return 0;
}
```

argc is the argument count

argv[1] holds the first number typed in at the command-line. Notice the atoi function.

Returning Values from Functions: passValue4.c

```
#include <stdio.h>
int add(int a, int b){ <-- Notice the return type
   int c;
   c = a + b; a=c; b=c;
   printf("\n Addition is : %d",c);
   return c; <-- Return value: c
}
int main(){
   int a, b, c;
   printf("\n Enter Any 2 Numbers : ");
   scanf("%d %d",&a,&b);
   printf("a is: %d, b is: %d\n", a, b);
   c = add(a, b); <--- Value returned from add stored in c
   printf("a is: %d, b is: %d\n", a, b);
   return 0;
}
```

Returning Values from Functions: passValue4.c

- Output:

```
Enter Any 2 Numbers : 5 6
a is: 5, b is: 6
Addition is : 11
a is: 5, b is: 6
```

Note: the values of `a` and `b` remained the same when accessed from function main. More about functions on later slides

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Arrays

- An array allows you to store many different values of same data type in a single unit

- Arrays are declared just like other variables, though the variable name ends with a set of square brackets
  - `char myName[50];` <---- You have seen this before
  - `int myVector[3];`
  - `int myMatrix[3][3];`

Arrays Example: arrayExample.c

```
#include <stdio.h>
int main(){
  int i;
  int age[4];
  age[0]=23;
  age[1]=34;
  age[2]=65;
  age[3]=74;
  for(i=0; i<4; i++){
    printf("age[%d]: %d\n", i, age[i]);
  }
  return 0;
}
```

Notice that count begins at 0

Output:

```
age[0]: 23
age[1]: 34
age[2]: 65
age[3]: 74
```

Structures

- Multiple variables can be combined into a single package called structure
- Members of the structure variable need not be of the same type
- They can be used to do database work in C! Example:

```
struct sample{
   int a;
   char b;
}
struct sample mySample;
```

- `typedef` is the keyword that can be used to simplify the usage of `struct`

```
typedef struct sample newType;
```

Structure Example: structExample.c

```
#include <stdio.h>

typedef struct point{
  double x;
  double y;
}point;

int main(){
  point myPoint;
  myPoint.x = 12.2;
  myPoint.y = 13.3;
  printf("X is %lf and Y is %lf\n",myPoint.x, myPoint.y);
  return 0;
}
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Pointers

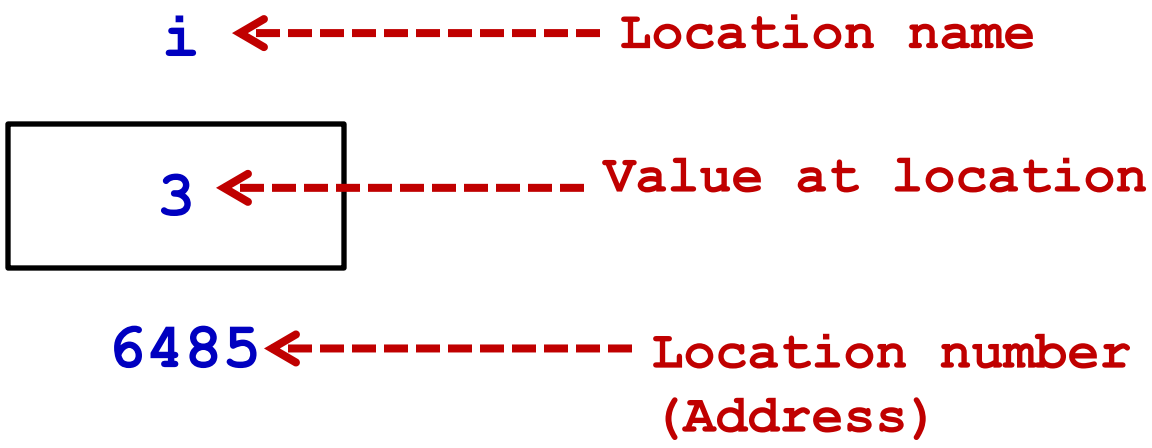
- A pointer is a variable that stores an address in memory - address of another variable
- For instance, the value of a pointer may be 42435. This number is an address in the computer's memory which is the start of some data
- We can dereference the pointer to look at or change the data
- Like variables, you have to declare pointers before you use them
- The data type specified with pointer declaration is the data type of the variable the pointer will point to

Revisiting Variable Declaration

- Consider the declaration

```
int i = 3;
```

- This declaration tells the C compiler to:
  - Reserve space in memory to hold the integer value
  - Associate the name `i` with this memory location
  - Store the value `3` at this location



‘Value at Address’ Operator: printAddress.c

```
#include <stdio.h>
int main(){
   int i=3;
   printf("\nAddress of i = %u", &i);
   printf("\nValue of i = %d", i);
   printf("\nValue of i = %d", *(&i));
   return 0;
}
```

& operator is ‘address of’ operator

* operator is ‘value at address of’ operator

Output:

```
Address of i = 2293532
Value of i = 3
Value of i = 3
```

Note:

&i returns the address of variable i

***(&i) returns the value at address of i**

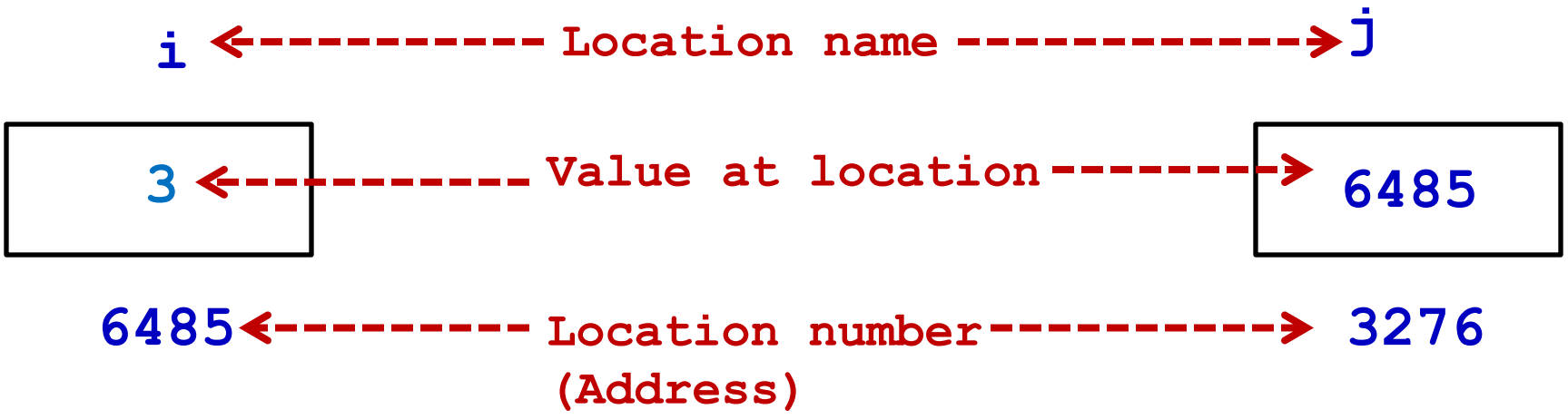
THE UNIVERSITY OF TEXAS AT AUSTIN

Pointer Expressions

- In the previous example, the expression **`&i`** returns the address of **`i`**.
- This address can be collected in a variable as

```
j = &i;
```

- `j` is a variable which contains the address of another variable and is declared as `int *j;`



Pointers: pointerExample2.c

```
#include <stdio.h>
int main(){
 int i=3;
 int *j;
 j = &i;
 printf("\nAddress of i = %u", &i);
 printf("\nAddress of i = %u", j);
 printf("\nAddress of j = %u", &j);
 printf("\nValue of j = %u", j);
 printf("\nValue of i = %d", i);
 printf("\nValue of i = %d", *(&i));
 printf("\nValue of i = %d", *j);
 return 0;
}
```

Output:

```
Address of i = 2293532
Address of i = 2293532
Address of j = 2293528
Value of j = 2293532
Value of i = 3
Value of i = 3
Value of i = 3
```


TACC
THE UNIVERSITY OF TEXAS AT AUSTIN

Key Concepts Related to Pointers

- Declaring a pointer

```
int *myIntPtr;
int* myIntPtr;
```

- Getting the address of a variable

```
int age = 3;
myIntPtr = &age;
```

- Dereferencing a pointer

```
*myIntPtr = 5;
```

Note: We just changed the value of age!

Pointers Example 2: ptrExample.c

```
#include <stdio.h>
int main(){
     int myValue;
     int *myPtr;
     myValue = 15;
     myPtr = &myValue;
     printf("myValue is equal to : %d\n", myValue);
     *myPtr = 25;
     printf("myValue is equal to : %d\n", myValue);
}
```

Output:

```
myValue is equal to : 15
myValue is equal to : 25
```


TACC
THE UNIVERSITY OF TEXAS AT AUSTIN

Pointers and Arrays

- The square-bracket array notation is a short cut to prevent you from having to do pointer arithmetic

```
char array[5];
array[2] = 12;
```

`array` is a pointer to **`array[0]`**

`array[2] = 12;` is therefore equivalent to

```
*(array+2) = 12;
```

Passing Address to Function: passValue3.c

```
#include <stdio.h>

void addUpdate(int *a, int *b){
   int c;
   c = *a + *b;
   printf("Addition is : %d\n",c);
   *a = c;
   *b = c;
}
int main(){
   int a, b;
   printf("Enter Any 2 Numbers : ");
   scanf("%d %d",&a,&b);
   printf("a is: %d, b is: %d\n", a, b);
   addUpdate(&a, &b);
   printf("a is: %d, b is: %d\n", a, b);
   return 0;
}
```

Notice the pointer

Notice &a, &b

Note: The values of **a** and **b** changed in **addUpdate** function .

Output of passValue3.c

- **Output:**

```
Enter Any 2 Numbers : 2 8
a is: 2, b is: 8
Addition is : 10
a is: 10, b is: 10
```

Dynamic Memory Allocation

- Dynamic allocation is the automatic allocation of memory at run-time
- It is accomplished by two functions: `malloc` and `free`
- These functions are defined in the library file `stdlib.h`
- `malloc` allocates the specified number of bytes and returns a pointer to the block of memory
- When the memory is no longer needed, the pointer is passed to `free` which deallocates the memory
- Other functions:
  - `calloc` allocates the specified number of bytes and initializes them to zero
  - `realloc` increases the size of the specified chunk of memory

Note: With arrays, static memory allocation takes place, that is at compile-time.

Example: dynMemAlloc.c (1)

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  int numStudents, avg, *ptr, i, sum = 0;
  printf("Enter the num of students :");
  scanf("%d",&numStudents);
  ptr=(int *)malloc(numStudents*sizeof(int));
  if(ptr == NULL){
    printf("\n\nMemory allocation failed!");
    exit(1);
  }
  for (i=0; i<numStudents; i++){
    printf("\nEnter the marks for the student %d\n", i+1);
    scanf("%d",(ptr+i));
  }
```

Example: dynMemAlloc.c (2)

```
    . . .
    for (i=0; i<numStudents; i++){
       sum = sum + *(ptr + i);
    }
    avg = sum/numStudents;
    printf("\nAvg marks = %d ",avg);
    return 0;
} // end of main function
```

```
Output:
Enter the num of students :3
Enter the marks for the student 1
10
Enter the marks for the student 2
20
Enter the marks for the student 3
30
Avg marks = 20
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Including Library File for Maths: mathExample.c

```
#include <stdio.h>
#include <math.h>
int main(){
 double myNum = 2.2;
 int times = 8 ;
 printf("Square root of %lf is: %lf\n",myNum, sqrt(myNum));
 return 0;
}
```

```
Output:
Square root of 2.200000 is: 1.483240
```

User-Defined Header Files

- Useful in multi-module, multi-person software development effort
- Save the following code in a file named head.h and don't compile/run it

```
/* This is my little header file named head.h */
#define HAPPY 100
#define SPIT printf
#define POOL {
#define PEEL }
```

User-Defined Header Files

- This is how the file head.h can be included in any program, here headTest.c

```
#include <stdio.h>
#include "head.h" <-- Notice the quotes around file name
int main()
POOL
SPIT("This guy is happy: %d percent\n", HAPPY);
return(0);
PEEL
```

```
Output:
This guy is happy: 100 percent
```

File I/O

- File pointer is required for accessing files to read, write or append

```
FILE *fp;
```

- **fopen** function is used to open a file and it returns a file pointer

```
FILE *fopen(const char *filename, const char *mode);
```

- The modes in which a file can be opened

```
r  - open for reading
w  - open for writing (file need not exist)
a  - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)
```

- To close a file

```
int fclose(FILE *a_file);
```

File I/O: fileExample.c

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int i, myInt;
  FILE *ifp;
  char *mode = "r";
  ifp = fopen("in.txt", mode);
  if (ifp == NULL) {
    fprintf(stderr, "Can't open input file in.txt!\n");
    exit(1);
  }else{
    for (i=0; i<10; i++){
      fscanf(ifp,"%d", &myInt);
      printf("%d\n",myInt);
    }
  }
  fclose(ifp);
  return 0;
}
```

Write to a File: writeToFile.c

```
#include <stdio.h>
int main(){
    FILE *fp;
    fp = fopen("in2.txt","a+");
    fprintf(fp,"\n%d",7000);
    fclose(fp);
    return 0;
}
```

Opening the file in append mode

fprintf is used for writing data to a file

References

- C Programming Language, Brian Kernighan and Dennis Ritchie
- Let Us C, Yashavant Kanetkar
- C for Dummies, Dan Gookin
- http://cplusplus.com