#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
Reading Quiz

Note the intimidating red border!
A variable is:

A. an area in memory that is reserved at run time to hold a value of particular type

B. an area in memory that is reserved at compile time to hold a value of particular type

C. an area in memory that is reserved when the variable is declared and assigned an initial value of zero
A variable is:

A. an area in memory that is reserved at run time to hold a value of particular type

B. an area in memory that is reserved at compile time to hold a value of particular type

C. an area in memory that is reserved when the variable is declared and assigned an initial value of zero
In C, a variable is declared by

A. using it in an arithmetic expression

B. preceding it with one or more keywords to indicate its ‘data type’, e.g. char c;

C. initializing it to a value
In C, a variable is declared by

A. using it in an arithmetic expression

B. preceding it with one or more keywords to indicate its ‘data type’, e.g. `char c;`

C. initializing it to a value
A function is:

A. A block of code that can be executed by invoking its name

B. any block of code within curly braces-{}

C. A block of code that can take multiple inputs as arguments and return multiple outputs
A function is:

A. A block of code that can be executed by invoking its name

B. any block of code within curly braces-{}

C. A block of code that can take multiple inputs as arguments and return multiple outputs
int foo (char , int );

A. is a call to execute the function ‘foo’

B. Is a declaration of the function ‘foo’ which takes two inputs and returns an ‘int’ value

C. Is an incorrect declaration because the keywords ‘char’ and ‘int’ are not followed by variable names
int foo (char , int );

A. is a call to execute the function ‘foo’

B. Is a declaration of the function ‘foo’ which takes two inputs and returns an ‘int’ value

C. Is an incorrect declaration because the keywords ‘char’ and ‘int’ are not followed by variable names
Variables declared within a function

A. can be used by other functions

B. can only be used within the function

C. continue to exist in memory as long as the program executes
Variables declared within a function

A. can be used by other functions

B. can only be used within the function

C. continue to exist in memory as long as the program executes
An arithmetic expression (e.g. 2*a+6)

A. Always evaluates to TRUE

B. Evaluates to TRUE if the result of the expression is non zero, (in this case if the variable ‘a’ is not equal to -3)

C. Evaluates to a number which is neither TRUE or FALSE
An arithmetic expression (e.g. $2*a+6$)

A. Always evaluates to TRUE

B. Evaluates to TRUE if the result of the expression is non zero, (in this case if the variable ‘a’ is not equal to -3)

C. Evaluates to a number which is neither TRUE or FALSE
What does a variable’s data-type specify?

A. The size of the variable and the interpretation of it value(s) when accessed

B. How long the variable persists in memory

C. Whether the variable stores integral or character values
What does a variable’s *data-type* specify?

A. The size of the variable and the interpretation of its value(s) when accessed

B. How long the variable persists in memory

C. Whether the variable stores integral or character values
A simple C program

```
#include <stdio.h>

main()
{
    int a=30;
    printf("CSE %d is fun!!",a); /* Printing a statement */
}
```

Basics of a C program:
- Program execution begins with `main()`
- Each statement terminates with a semi colon
- Libraries used by including a header file
- Comments are added using: `/* */`
C Programming

- Procedural thought process
- No built-in object abstractions
  - data separate from methods/functions
- Low memory overhead compared to Java
  - No overhead of classes
  - Relatively fast
- Heap memory management manual
- Pointers to manipulate shared data
Key C concepts allow us to:

- Manipulate data-objects
- Control the flow of program execution
- Break large computing tasks into smaller ones
Manipulating data objects in C
Different types of data

- Data differ in a number of ways:
  - Constant vs. variable
  - Basic vs. structured (derived)
  - Data type: int, float, char . . .

Data in a C program also differs based on scope and lifetime (more later)
How we manipulate variables depends on their *data-type*

Data-types

- Basic: int, char, float
  - integer data types used in arithmetic operations but not char data types

- Derived: Pointers, Arrays, Structures, enumerations
  - Derived types are accessed and manipulated differently from basic types
Basic data object in memory

A region in memory that contains a value and is associated with a name/identifier.

Attributes of a Data-object/variable:
- Name/identifier
- Value
- Address: Location in memory
- Size
- A unique data-type
- Lifetime
- Scope
What attributes of a variable are specified in a declaration statement?

A. Name, value, location, data type, lifetime, scope

B. Name, data-type, value and scope

C. Name, data-type, scope; its value may or may not be specified

D. Name, data-type; its value, lifetime and scope may or may not be specified
What attributes of a variable are specified in a declaration statement?

A. Name, value, location, data type, lifetime, scope

B. Name, data-type, value and scope

C. Name, data-type, scope; its value may or may not be specified

D. Name, data-type; its value, lifetime and scope may or may not be specified
Declarations and definitions

- `char c='a';`  /* 1 byte */
- `short s;`  /* 2 bytes */
- `int a;`  /* usually 4 bytes - signed */
- `unsigned int a=0;`  /* usually 4 bytes*/
- `float f;`  /* usually 4 bytes use sizeof(float)*/

Scope and lifetime are often implicit but sometimes we have to use specific keywords:

- `static int a=0;`  /* Defines lifetime*/
- `extern int a;`  /* Extends scope to multiple files*/
Which of the following statements is true after a variable is “declared”?

A. We can access and change both its value and location

B. We can access and change only its value

C. We can access its value and location but can only change its value
Which of the following statements is true after a variable is “declared”?

A. We can access and change both its value and location
B. We can access and change only its value
C. We can access its value and location but can only change its value
Accessing value and location

- To access/change the value of a basic type:
  - Use the variable name
  - Is there another way?

```c
y = x;
x = 10;
y = x > y ? x : y;
```

- To access the location/address, use the address operator ‘&’

```c
&x  (is 102)
```
Type casting
Promotion and truncation

- Often arithmetic expressions contain mixed integral types
- Promotion: In an expression all variables are ‘promoted’ to the data type of the largest size
  ```
  char c=4;
  int i = c+ 10; /* c is promoted to an int */
  /*What is the value in i? */
  ```
- Truncation: If the value of a larger data type is assigned to a smaller one, the compiler drops the most significant bits
  ```
  short i = 0x0104;
  char c =i; /*The value in i is truncated*/
  /* c =0x04 – Least significant byte is assigned */
  ```
Type casting

• If we want the variables in an expression to be interpreted differently, use casting

• Example:
  ```
  int i=10;
  float f = i/3;
  ```

• In this case, f=3.0

• If we don’t want the result of the division to get truncated to an int, use explicit casting

  ```
  int i=10;
  float f = (float)i/3; /* Explicit cast of int to float */
  ```
Functions:
Breaking large computing tasks to smaller ones
C Programming

• Procedural thought process

main () /* High level Outline */
{
  ...
  get_input(arg1) /*Comment: Step 1 */
  perform_step_2(arg2);
  perform_step_3();
  store_result(); /* Print output or store in a file */
}
Overview of Functions

Functions make code easy to

- Maintain
- Debug
- Reuse
Functions Overview

void swap(int x, int y); //Declaration

void swap(int x, int y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}

When swap is called
- 3 local variables are created: x, y, tmp
- the value in x and y are interchanged
Functions: Call by value

main() {
  ...
  swap(a, b);
  ...
}

Q: Are the value of variables ‘a’ and ‘b’ interchanged after `swap` is called?

A. Yes, because that’s what is implemented by the ‘swap’ routine

B. No, because the inputs to swap are only copies of ‘a’ and ‘b’
Functions: Call by value

```c
main() {
    . . .
    swap(a, b);
    . . . .
}
```

Q: Are the value of variables ‘a’ and ‘b’ interchanged after `swap` is called?

A. Yes, because that’s what is implemented by the ‘swap’ routine

B. No, because the inputs to `swap` are only copies of ‘a’ and ‘b’
The C runtime environment
What does gcc do?

% gcc hello.c
What does gcc do?

% gcc hello.c

*Command prompt (in linux/unix systems)*
What does gcc do?

% gcc hello.c

*Input to gcc: C program*
What does gcc do?

% gcc hello.c

Diagram:

- hello.c
- gcc
- a.out
What does gcc do?

% gcc hello.c

“Source”
Program in C

```
#include <stdio.h>
void func1(int a, char *b)
{
    if(a > 0)
    {
        *b = ‘a’ ;
    }
}
int main()
{
    ...
    func1();
    printf(“\abc”);
}
```
What does gcc do?

```
#include <stdio.h>
void func1(int a, char *b)
{
    if(a > 0)
    { *b = 'a'; }
} 
int main()
{
    ..... 
    func1();
    printf("\abc");
}
```

% gcc hello.c

**“Source”**

Program in C

hello.c

**“Executable”: Equivalent program in machine language**

```
0000 1001 1100 0110
1010 1111 0101 1000
1010 1111 0101 1000
0000 1001 1100 0110
1100 0110 1010 1111
0101 1000 0000 1001
0101 1000 0000 1001
1100 0110 1010 1111
```
What does gcc do?

% gcc hello.c
% ./a.out (executable loaded in memory and processed)
Also referred to as “running” the C program

“Source”: Program in C

“Executable”: Equivalent program in machine language
How is ‘other’ code included?

- Include Header files (.h) that contain function declarations - the function interface
- The corresponding .c files contain the actual code

```c
#include <stdio.h>

void func1(int a, char *b)
{
    if(a > 0)
    {
        *b = 'a';
    }
}

int main()
{
    ..... 
    func1();
    printf("\abc");
}
```
What does the executable contain?

• Instructions or “code”
• Data

What is data?

• Any information that instructions operate on (objects in memory)
C Runtime Environment

- **“Code”** (instructions in machine language)
- **“Data”** (initialized and uninitialized - static allocated)

Both code and data don’t change in size

- **“Heap”** (for dynamically allocated data)
- **“Stack”** (for function local variables)

Heap and stack change in size as the program executes
C Runtime Environment

- Code
- Initialized Data
- Uninitialized Data
- Heap
- Stack
Different types of data

What factor differentiates the following three ‘categories’ of data in a C program:

• Global and static variables
• Local variables
• Dynamic variables

A. Scope
B. Lifetime
C. Representation
Different types of data

What factor differentiates the following three ‘categories’ of data in a C program:

• Global and static variables
• Local variables
• Dynamic variables

A. Scope
B. Lifetime
C. Representation
Data differing in lifetime

• Lifetime: The interval between time of creation and end of existence
Possible lifetimes of data

1. **Lifetime = Execution time of program**
   - Initialized or uninitialized
   - Must be indicated in the executable
   - Space is allocated with start of execution (static allocation)

   How much space indicated in executable
Possible lifetimes of data

1. Lifetime = Execution time of program

2. Lifetime = Time between explicit creation and explicit deletion
   - Programmer dynamically allocates memory
   - In C new data created when malloc() is called and destroyed when free() is called
   - Space in memory created *during* program execution (Heap allocation)
     - How much data will be created on the heap is not known ahead of time
     - This type of data cannot be indicated in the executable ahead of time
Possible lifetimes of data

1. Lifetime = Execution time of program
2. Lifetime = Time between explicit creation and explicit deletion
3. Lifetime = Execution time of a function (time between function call and function return)
   - Local variables of functions, parameters of function
   - Space is allocated when the function execution starts and is reclaimed on return from function (stack allocation)
     - Stack: data structure like a pile of books
     - Last in First Out: Adding on top (push), removing from top (pop)
Stack Allocation: Function local variables and parameters

- When program execution starts

What if main calls the function func()?
Stack Allocation: Function local variables and parameters

- Variables whose lifetime is the execution time of function are managed using the stack structure.

Local variables of main()