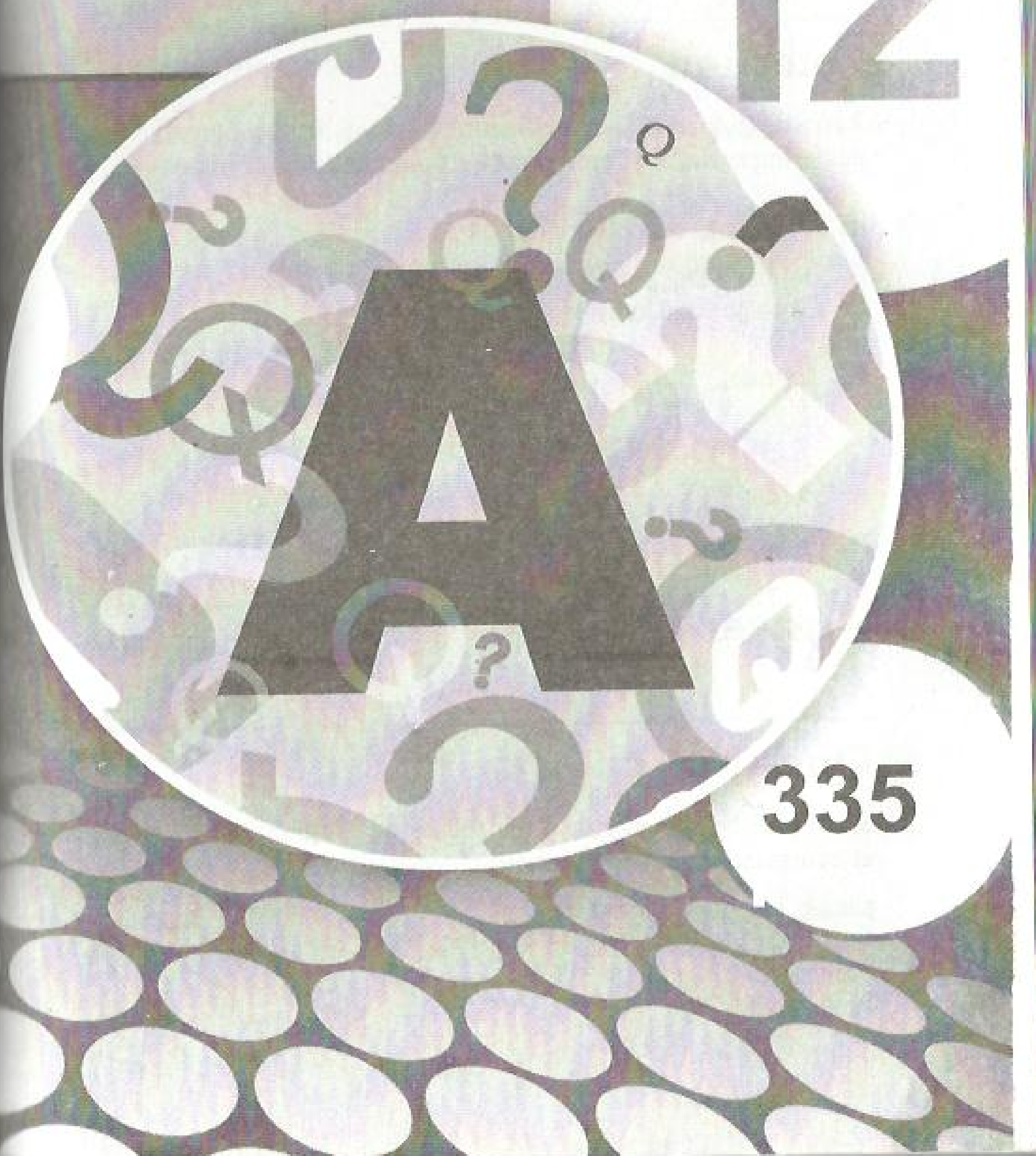
Command Line Arguments

12



Question 12.1

What do the 'c' and 'v' in *argc* and *argv* stand for?

Answer

Count of arguments and vector (array) of arguments.

Question 12.2

According to ANSI specifications which is the correct way of declaring *main()* when it receives command-line arguments?

A.
```
int main ( int argc, char *argv[ ] )
```

B.
```
int main ( argc, argv )
 int argc ; char *argv[ ] ;
```

C.
```
int main( )
{
    int argc ; char *argv[ ] ;
}
```

D. None of the above

Answer

A

Question 12.3

What will be the output of the following program if it is executed at command-line as shown below?

```
sample
```

```
/* sample.c */
#include <stdio.h>
```

```
int main ( int argc, char **argv )
{
    printf ( "%s\n", argv[argc - 1] ) ;
    return 0 ;
}
```

Answer

c:\sample\Debug\sample.exe

On execution in Visual Studio we get complete path of the file "sample.exe".

In gcc we get the executable code would get created in either "a.out" or "sample.out". Assuming it is in "sample.out", we can execute it as follows:

```
$./sample.out
```

This would produce the output

```
./sample.out
```

Question 12.4

If different command-line arguments are supplied at different times would the output of the following program change? [Yes/No]

```
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%d\n", argv[ argc ] ) ;
    return 0 ;
}
```

Answer

No

Question 12.5

What will be the output of the following program?

```
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%d\n", argv[ argc ] + 5 ) ;
    return 0 ;
}
```

Answer

5

Question 12.6

What will be the output of the program (*myprog*) given below if it is executed from the command-line as

```
myprog 10 20 30
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    int i ;
    for ( i = 0 ; i < argc ; i++ )
        printf ( "%s ", argv[ i ] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

Answer

In Visual Studio we get the output as:

```
c:\myprog\Debug\myprog.exe 10 20 30
```

In gcc we get the output as:

```
./myprog.out 10 20 30
```

Question 12.7

What will be the output of the program (*myprog*) if it is executed from the command-line as shown below?

```
myprog 1 2 3
```

```
/* myprog.c */
#include <stdio.h>
#include <stdlib.h>
int main ( int argc, char *argv[ ] )
{
    int i, j = 0 ;
    for ( i = 0 ; i < argc ; i++ )
        j = j + atoi ( argv[ i ] ) ;

    printf ( "%d\n", j ) ;
    return 0 ;
}
```

A. 123
B. 6
C. Error
D. "123"

Answer

B. When *atoi( )* tries to convert *argv[0]* to a number it cannot do so (*argv[0]* being a file name) and hence returns a zero.

Question *12.8*

What will be the output of the program given below (*myprog*) if it is executed from the command-line as shown below?

```
myprog one two three
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%s\n", *++argv ) ;
    return 0 ;
}
```

Answer

one

Question *12.9*

State which of the following statements is True or False about command-line arguments?

A. Every time we supply new set of values to the program at command prompt, we need to recompile the program.

B. Even if integer/float arguments are supplied at command prompt they are treated as strings.

C. The first argument to be supplied at command-line must always be the count of total arguments.

Answer

A. False
B. True
C. False

Question *12.10*

Suppose the following program (*myprog*) is executed as shown below:

```
myprog one two three
```

If the first value that it prints is 65517, what will be the rest of the output?

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    int i ;
    for ( i = 1 ; i <= 3 ; i++ )
        printf ( "%u ", &argv[ i ] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

Answer

65521 65525

Question *12.11*

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog one two three
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%c\n", **++argv ) ;
    return 0 ;
}
```

Answer

o

Question 12.12

The maximum combined length of the command-line arguments including the spaces between adjacent arguments is

A. 128 characters
B. 256 characters
C. 67 characters
D. It may vary from one operating system to another

Answer

D

Question 12.13

What will be the output of the following program?

```
#include <stdio.h>
int main ( int argc, char *argv[ ], char *env[ ] )
{
    int i ;
    for ( i = 1 ; i < argc ; i++ )
        printf ( "%s \n", env[ i ] ) ;
```

```
    return 0 ;
}
```

A. List of all environment variables
B. List of all command-line arguments
C. Count of command-line arguments
D. Error: Cannot have more than 2 arguments in main( )

Answer

A

Question 12.14

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog one two three
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    int i ;
    for ( i = 1 ; i <= 3 ; i++ )
        printf ( "%c", *argv[ i ] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

Answer

Ott

Question 12.15

Which of the following is the correct output of the code snippet given below if the program is executed as sample "*.c"?

```
/* sample.c */
#include <stdio.h>
int main ( int argc, int *argv )
{
    int i ;
    for ( i = 1 ; i < argc ; i++ )
        printf ( "%s ", argv[ i ] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

A. *.c
B. "*.c"
C. myprog *.c
D. List of all files and folders in the current directory.

Answer

A

Question 12.16

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog one two three
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    int i = 0 ;
```

```
    i += strlen ( argv[ 1 ] ) ;
    while ( i > 0 )
    {
        i-- ;
        printf ( "%c", argv[ 1 ][ i ] ) ;
    }
    printf ( "\n" ) ;
    return 0 ;
}
```

Answer

eno

Question 12.17

In Turbo C/C++ under DOS if we want that any wildcard characters in the command-line arguments should be appropriately expanded, are we required to make any special provision? If yes, which? How is this feature handled by Linux and Unix?

Answer

Yes. Compile the program as

```
tcc myprog wildargs.obj
```

This compiles the file *myprog.c* and links it with the wildcard expansion module WILDARGS.OBJ, then run the resulting executable file MYPROG.EXE

If you want the wildcard expansion to be default so that you won't have to link your program explicitly with WILDARGS.OBJ, you can modify your standard C?.LIB library files to have WILDARGS.OBJ linked automatically. To achieve this we have to remove SETARGV from the library and add WILDARGS. The following commands will invoke the Turbo Librarian to modify all the standard library files (assuming the current directory contains the standard C libraries, and WILDARGS.OBJ):

```
tlib cs -setargv +wildargs
tlib cc -setargv +wildargs
tlib cm -setargv +wildargs
tlib cl -setargv +wildargs
tlib ch -setargv +wildargs
```

Under Linux and Unix the shell expands the wildcard characters before passing them to *main( )*. For example, if we execute the program as shown below

```
myprog *.c
```

then all filenames with extension ".c" would be passed to *main( )* as command-line arguments.

Question *12.18*

Does there exist any way to make the command-line arguments available to other functions without passing them as arguments to the function? [Yes/No]

Answer

Yes. Using the predefined variables *_argc*, *_argv*. This is a compiler dependent feature. It works in TC/TC++ but not in gcc or Visual Studio.

Question *12.19*

What will be the output of the following program (*myprog*) if it is executed from the command-line as

```
myprog Jan Feb Mar
```

```
/* myprog.c */
```

```
#include <dos.h>
#include <stdio.h>
int main( )
{
    int i ;
    for ( i = 1 ; i < _argc ; i++ )
        printf ( "%s ", _argv[i] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

Answer

Jan Feb Mar

This is a compiler dependent feature. It works in TC/TC++ but not in gcc or Visual Studio.

Question *12.20*

If the following program (*myprog*) is present in the directory *c:\bc\tucs* then what will be the output on its execution?

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%s \n", argv[0] ) ;
    return 0 ;
}
```

A. MYPROG
B. C:\BC\TUCS\MYPROG
C. Error
D. C:\BC\TUCS

Answer

B

Question 12.21

Which is an easy way to extract *myprog* from the output of program 12.20 above?

Answer

```
#include <dir.h>
int main ( int argc, char *argv[ ] )
{
    char drive[ 3 ], dir[ 50 ], name[ 8 ], ext[ 3 ] ;
    printf ( "%s\n", argv[ 0 ] ) ;
    fnsplit ( argv[ 0 ], drive, dir, name, ext ) ;
    printf ( "%s\n%s\n%s\n%s\n", drive, dir, name, ext ) ;
    return 0 ;
}
```

This is a compiler dependent feature. It works in TC/TC++ but not in gcc or Visual Studio.

Question 12.22

Which of the following is true about *argv*?

A. It is an array of character pointers
B. It is a pointer to an array of character pointers
C. It is an array of strings
D. None of the above

Answer

A

Question 12.23

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog monday tuesday wednesday thursday
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    while ( --argc > 0 )
        printf ( "%s ", *++argv ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

A. myprog monday tuesday wednesday thursday
B. monday tuesday wednesday thursday
C. myprog tuesday thursday
D. None of the above

Answer

B

Question 12.24

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog friday tuesday sunday
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%c\n", ( *++argv )[ 0 ] ) ;
    return 0 ;
}
```

A. m
B. f
C. myprog
D. Friday

Answer

B

Question 12.25

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog friday tuesday sunday
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%c\n", **++argv ) ;
    return 0 ;
}
```

A. m
B. f
C. myprog
D. Friday

Answer

B

Question 12.26

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog friday tuesday sunday
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%c\n", *++argv[1] ) ;
    return 0 ;
}
```

A. r
B. f
C. m
D. y

Answer

A

Question 12.27

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog friday tuesday sunday
```

```
/* myprog.c */
#include <stdio.h>
int main ( int sizeofargv, char *argv[ ] )
{
    while ( sizeofargv )
        printf ( "%s ", argv[--sizeofargv] ) ;
    printf ( "\n" ) ;
    return 0 ;
}
```

A. myprog friday tuesday sunday
B. myprog friday tuesday
C. sunday tuesday friday myprog
D. sunday tuesday Friday

Answer

C

Question *12.28*

Which of the following statements are correct about the code given below:

```
int main ( int ac, char *av[ ] )
{
}
```

A. *ac* contains count of arguments supplied at command-line.
B. *av[ ]* contains addresses of arguments supplied at command-line.
C. In place of *ac* and *av*, *argc* and *argv* should be used.
D. *ac* contains address of an integer whose value is equal to the count of arguments supplied at command-line.
E. The variables *ac* and *av* are always local to *main( )*.
F. The maximum combined length of the command-line arguments including the spaces between adjacent arguments should be 128 characters.

Answer

A. True
B. True

C. False
D. False
E. True
F. False

Question *12.29*

What will be the output of the following program (*myprog*) if it is executed from the command-line as shown below?

```
myprog Good Morning
```

```
/* myprog.c * /
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%d %s\n", argc, argv[1] ) ;
    return 0 ;
}
```

Answer

3 Good

Question *12.30*

What will happen if the following program (*myprog*) is executed from the command-line as shown below?

```
myprog 1 2 3
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    int j ;
    j = argv[ 1 ] + argv[ 2 ] + argv[ 3 ] ;
    printf ( "%d\n", j ) ;
    return 0 ;
}
```

Answer

The code will causes an error at runtime as we are attempting to add addresses in the expression

```
j = argv[ 1 ] + argv[ 2 ] + argv[ 3 ] ;
```

Question *12.31*

What will be the output of the program given below?

```
#include <stdio.h>
void func ( int ) ;

int main ( int argc )
{
    printf ( "%d ", argc ) ;
    func ( argc ) ;
    return 0 ;
}

void func ( int i )
{
    if ( i != 4 )
        main ( ++i ) ;
}
```

Answer

```
1 2 3 4
```

Question *12.32*

What will be the output of program given below for the following command-line arguments?

```
myprog  hi good morning
myprog hello god morning
myprog good morning
```

```
/* myprog.c */
#include <stdio.h>
int main ( int argc, char *argv[ ] )
{
    printf ( "%s\n", argv[ 0 ] ) ;
    return 0 ;
}
```

Answer

Irrespective of command-line arguments the program would always output "c:\myprog\Debug\myprog.exe" in case of Visual Studio and "./myprog.out" in case of gcc.

Question *12.33*

To correctly add three integers supplied at command-line which statement will you add to the code given below?

```
#include <stdio.h>
#include <stdlib.h>
int main ( int argc, char *argv[ ] )
{
    int i, j = 0 ;
    for ( i = 0 ; i < argc ; i++ )
    {
        /* add suitable statement here */
    }
```

```
        printf ( "%d\n", j ) ;
        return 0 ;
}
```

Answer

```
j = j + atoi ( argv[ i ] ) ;
```

Question 12.34

To correctly add three floats supplied at command-line which statement will you add to the code given below?

```
#include <stdio.h>
#include <stdlib.h>
int main ( int argc, char *argv[ ] )
{
    int i ;
    float j = 0.0 ;
    for ( i = 0 ; i < argc ; i++ )
    {
        /* add suitable statement here */
    }
    printf ( "%f\n", j ) ;
    return 0 ;
}
```

Answer

```
j = j + atof ( argv[ i ] ) ;
```

Question 12.35

Match the following if we execute a "sample" program with command-line arguments as shown below:

```
sample PR1.C PR2.C
```

argc	2
argv[0]	3
argv[1]	Base address of the string "PR1.C"
argv[2]	Base Address of "sample"
	Base address of the string "PR2.C"

Answer

argc	3
argv[0]	Base Address of "sample"
argv[1]	Base address of the string "PR1.C"
argv[2]	Base address of the string "PR2.C"

```
    printf ( "%d\n", j ) ;
    return 0 ;
}
```

Answer

```
j = j + atoi ( argv[ i ] ) ;
```

Question 12.34

To correctly add three floats supplied at command-line which statement will you add to the code given below?

```
#include <stdio.h>
#include <stdlib.h>
int main ( int argc, char *argv[ ] )
{
    int i ;
    float j = 0.0 ;
    for ( i = 0 ; i < argc ; i++ )
    {
        /* add suitable statement here */
    }
    printf ( "%f\n", j ) ;
    return 0 ;
}
```

Answer

```
j = j + atof ( argv[ i ] ) ;
```

Question 12.35

Match the following if we execute a "sample" program with command-line arguments as shown below:

```
sample PR1.C PR2.C
```

argc	2
argv[0]	3
argv[1]	Base address of the string "PR1.C"
argv[2]	Base Address of "sample"
	Base address of the string "PR2.C"

Answer

argc	3
argv[0]	Base Address of "sample"
argv[1]	Base address of the string "PR1.C"
argv[2]	Base address of the string "PR2.C"