Diploma in

Computer Science

Variable Arguments

Objectives

- Recall the role of arguments in functions
- Comprehend the concept of variable arguments
- Explain arguments in a program
- Use the functions contained in the stdarg header file

Arguments

A step back…

Arguments take two forms:

- Actual arguments
- Formal arguments

Formal parameters

Variables declared in function prototype or definition

```
Int calculation(int x, int y);
```

Example

Use numbers, expressions, or function calls as actual parameters

Arguments

Not in the form of declarations

Implemented in the call to the function

Passing arguments to functions

Methods of passing arguments

Two ways:

- As a value: a copy is passed on to the function
- As an address: function has direct access

Variadic functions

Arguments in varying quantities

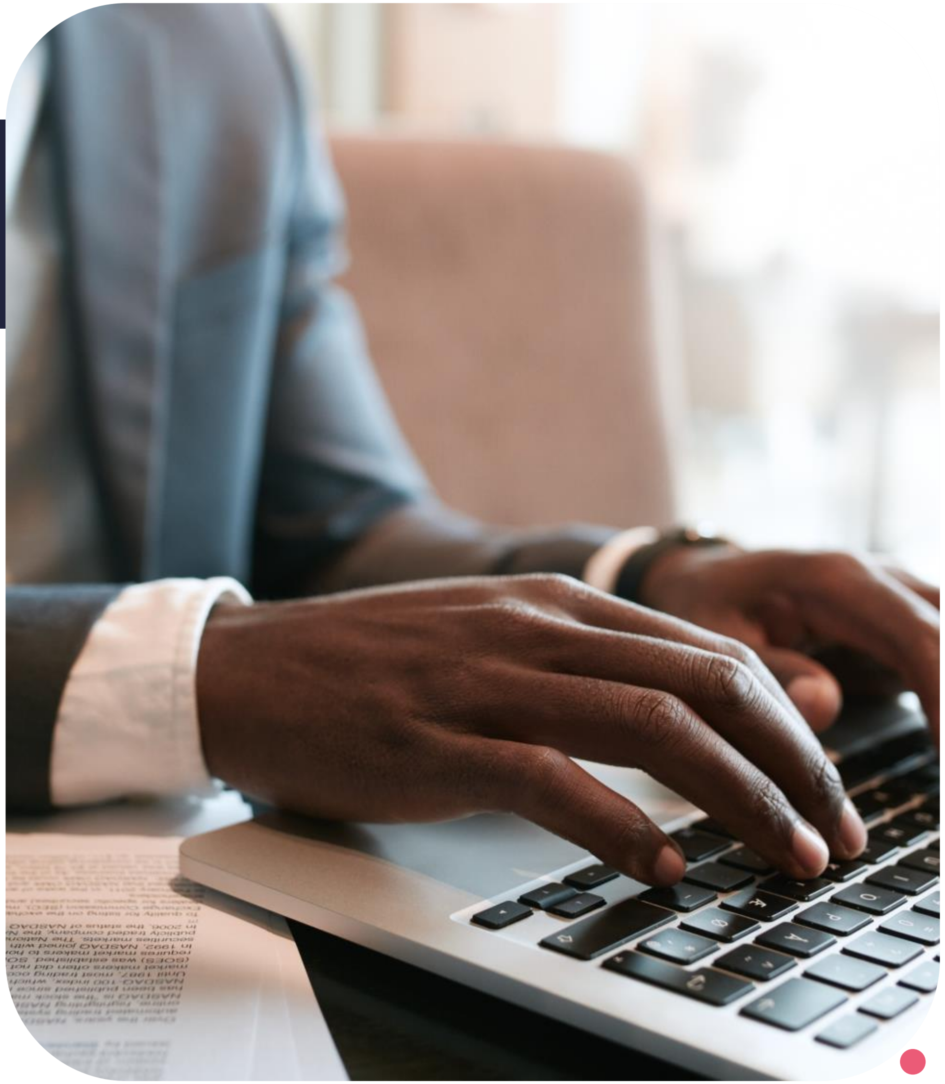
- Once code is compiled, arguments cannot be modified
- Don’t know how much data will be encountered at runtime
- With C, can send a variable number of arguments to a function
- Not an issue if you roughly know number of elements

Why do we need variadic functions?

When you define an ordinary function, you need to supply a specific number of arguments.

Variable arguments

- Some functions expect a varying number of arguments
- Variadic functions solve this by allowing you to supply the function with a variable number of arguments
- Have one function that creates any number of array elements
- Function becomes a variadic function



Example: printf function

- Can handle just about any amount of information
- Anything between the parentheses is an argument
- Gets passed to the printf function and info is displayed on screen

Example: printf function

```
int a=0, b=1, c=2;
printf(“%d , %d ,%d”, a,b,c);
printf(“%d ,%d”, a,c);
```

Example

Both arguments are syntactically correct.

Example: scanf function

```
scanf(“%d”, amount);
```

Example

Variable arguments offer greater flexibility.

Variadic functions

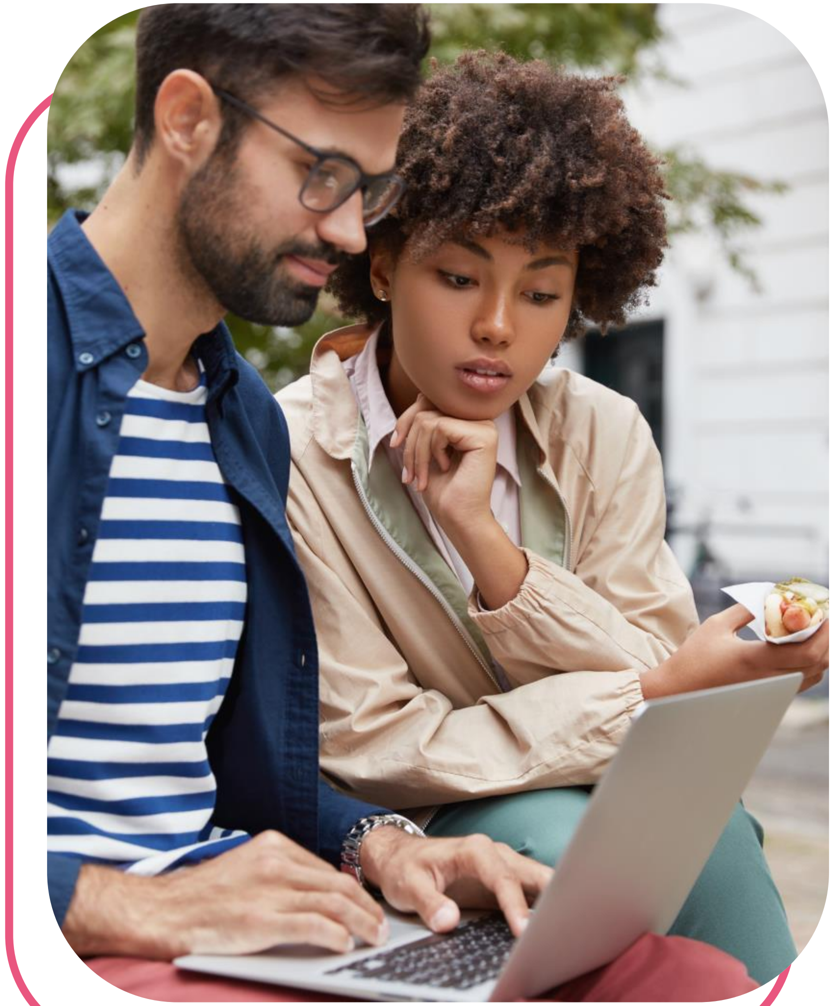
```
int functionname (int firstargument, ...);
```

Syntax

Variadic function

```
 #define __va_argsiz(t) (((sizeof(t) + sizeof(int) -
   1) / sizeof(int)) * sizeof(int))
// several lines are skipped
#if defined __GNUC__ && __GNUC__ >= 3
Typedef __builtin_va_list va_list;
#else
Typedef char* va_list;
#endif
// several lines are skipped
#ifdef __GNUC__
#define va_start ( ap, pn ) ((ap) = ((va_list)
   __builtin_next_arg(pn)))
#else
#define va_start ( ap, pn ) ((ap) = ((va_list) (&pn)
   + __va_argsiz(pn)))
#endif
// several lines are skipped
#define va_arg(ap, t) (((ap) = (ap) +
   __va_argsiz(t)), *((t*) (void*) ((ap) -
   __va_argsiz(t))))
// several lines are skipped
#define va_end(ap) ((void)0)
```

Example



vfprintf function

Three macros and one special data type are used to implement the variadic functions, namely va_start, va_arg, va_end and va_list.

Variadic function components

typedef va_list: list of variable arguments

char*: initialised by the address of the first variable argument

Holds information about next va_arg()

Computer gets all subsequent argument values by calling va_arg() repeatedly

va_list: tracks progression

va_start(): initialises the variable va to point to variable argument after count

va_start(va_list ap, pn);

Takes a va_list variable, ap, and the last-named parameter, pn, as its input

Initialises ap by adding the size of pn to its address

After calling va_start, ap points to the address of first unnamed parameter

va_start(va_list ap, pn);

Takes the va_list variable ap and the type of the next unnamed parameter T as input

Returns the value of the next unnamed parameter

If the va_arg()function is invoked, and there are no more arguments, it results in undefined behaviour

Standard argument header: stdarg...

...is used to write programs that accept an indefinite number of arguments

Indefinite arguments

- Size of the unnamed argument list is generally an unknown value
- No reliable way to pass unnamed arguments into another variadic function
- Number and size of arguments passed into that call will still need to be known at compile time

- Most standard libraries provide alternative way to access the unnamed argument list

  Example: vfprintf()

- Providing variadic API functions without also providing equivalent functions accepting va_list instead is considered bad programming practice

Indefinite arguments

Variable argument pitfalls

- Compiler usually cannot check argument types
- Typecasting conversions should be considered
- Type promotions and demotions will also result in undefined behaviour
- Receiving function must expect the new types

Did you know?

Hello World! exposes some of the most dangerous features of the language.

Our project

Planning

- Start with problem statement
- Plan solution with series of steps

Step 1: Build a solution

- Write down ways to solve the problem
- Flesh out the probable solution
- Think of strategy that the program will follow

Step 2: Write pseudocode

- Gives flexibility when strategising how the program will work
- Provides rough idea for flow chart

Step 3: Draw the flow chart

- Program must be in logical state
- Flow chart needs to be as detailed as possible
- Stick to pseudocode for consistency

Step 4: Write actual code

- Stick to pseudocode for guidance
- Keep visualisation of completed solution in mind

More about our project....

- Will be on a system that accepts student details
- Calculates average marks and totals
- Menu for user to pick from
- Fully-functional program