CS2211b

Software Tools and Systems Programming

Week 6b

Introducing C

To complete your Midterm Check-In, please visit:

feedback.uwo.ca

your Feedback

Western Science

Western

Announcements

No Office Hours or Labs on Reading Week

Quiz #2 Today

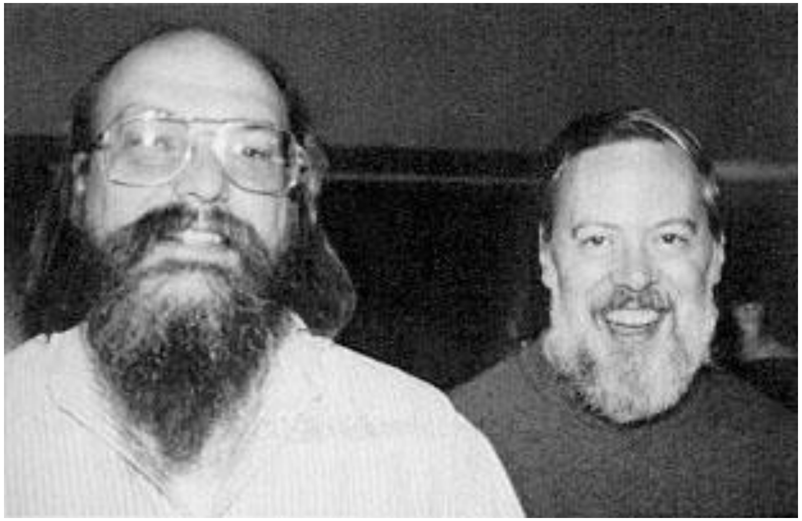
Midterm

- Saturday March 3rd @ 9:30 AM
- Location: WSC 55
- Length: 2 hours
- Content: Everything up to C (today)
- Format: Mixed (True/False, Multiple Choice, Short Answer, Long Answer)
- Study questions to be posted
- More details on the 27th

Introducing C: History, Standardization & Strengths

History of C

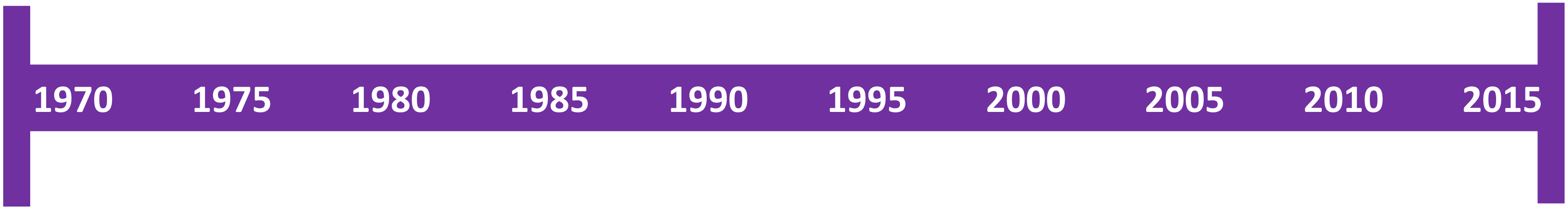
- By product of the UNIX operating system.
- Developed at AT&T's **Bell Laboratories** by **Ken Thompson**, **Dennis Ritchie**, and others.
- Was created for the purpose of rewriting the UNIX operating system in a more portable high-level language.
- Aided in UNIX's popularity and widespread adoption.



Thompson **Ritchie**

History of C

Timeline



1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

History of C

Timeline

1969

Creation of UNIX

Ken Thompson creates the first version of UNIX in assembly for the DEC PDP-7 minicomputer.

DEC PDP-7 Minicomputer

$72,000US (equivalent to $559,121US in 2017)

1970
1975
1980
1985
1990
1995
2000
2005
2010
2015

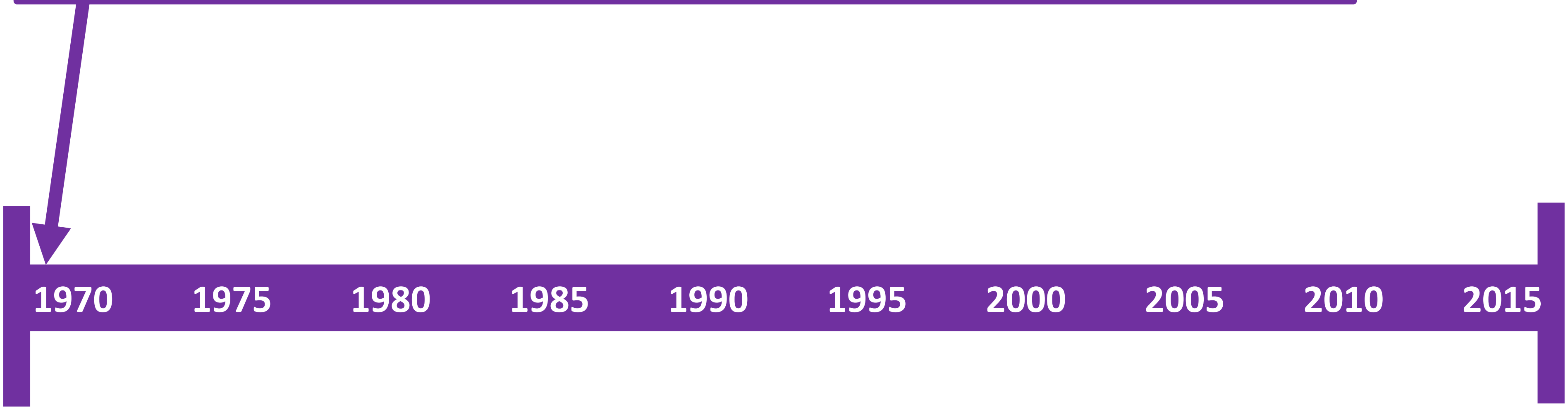
History of C

Timeline

1969

Creation of B

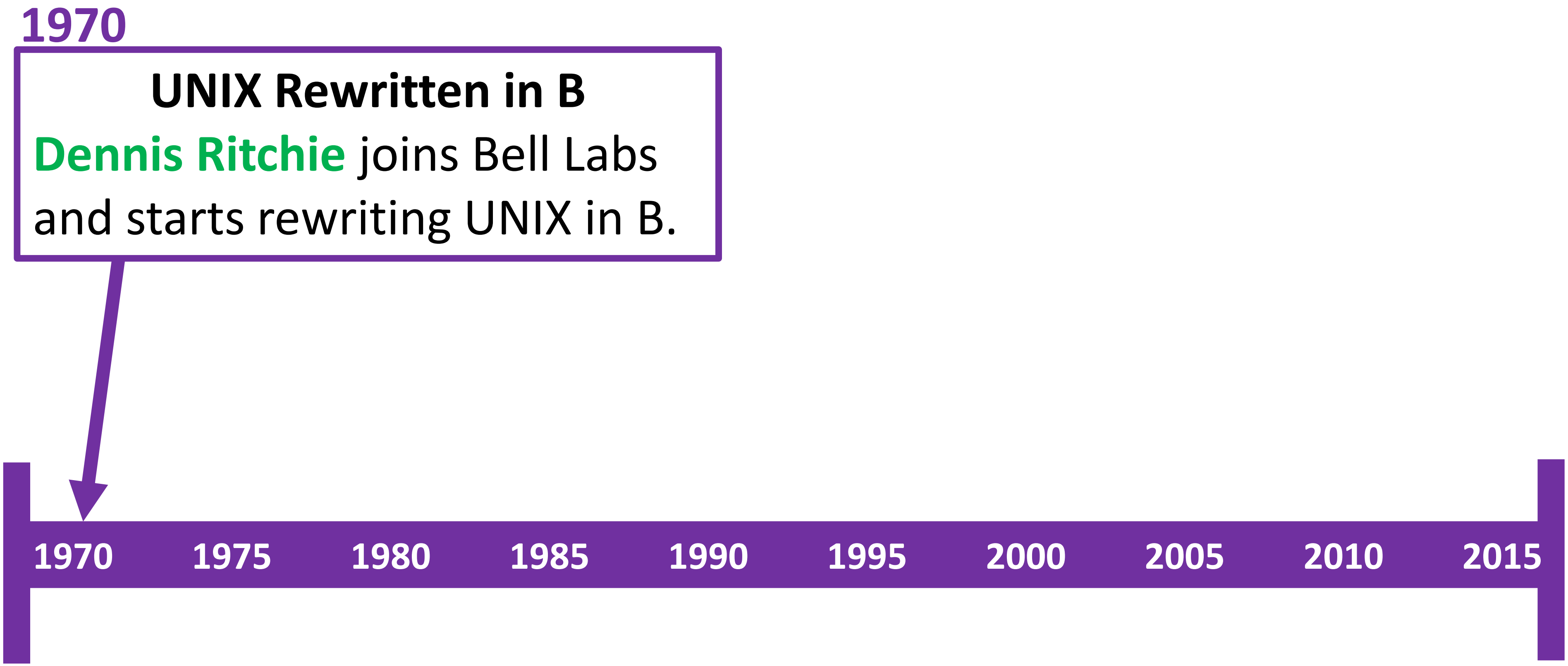
Ken Thompson creates the **B programming language** based on BCPL. B was a striped down version of BCPL to fit within the memory limits of microcomputers of the time.



1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

History of C

Timeline



1970

UNIX Rewritten in B

Dennis Ritchie joins Bell Labs and starts rewriting UNIX in B.

History of C

Timeline

1971 - 1972

“New B”

The typeless nature of the B programming language became an issue on new hardware like the PDP-11 minicomputer Bell Labs had recently acquired. **Ritchie** starts creating an extended version of B called “New B” at the time to add new features.

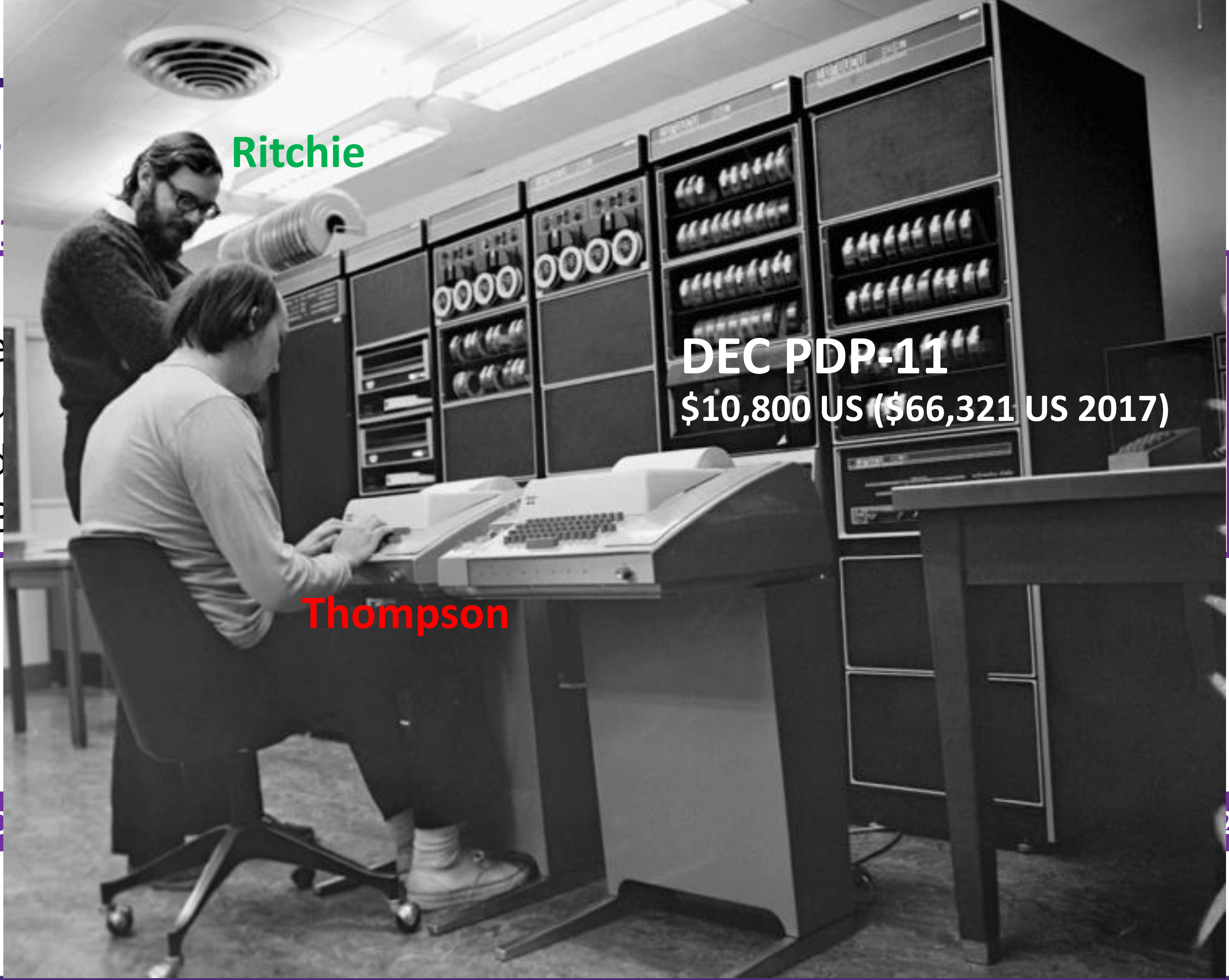
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015

H

T

197

The
issu
had
of E



DEC PDP-11

$10,800 US ($66,321 US 2017)

1970

2015

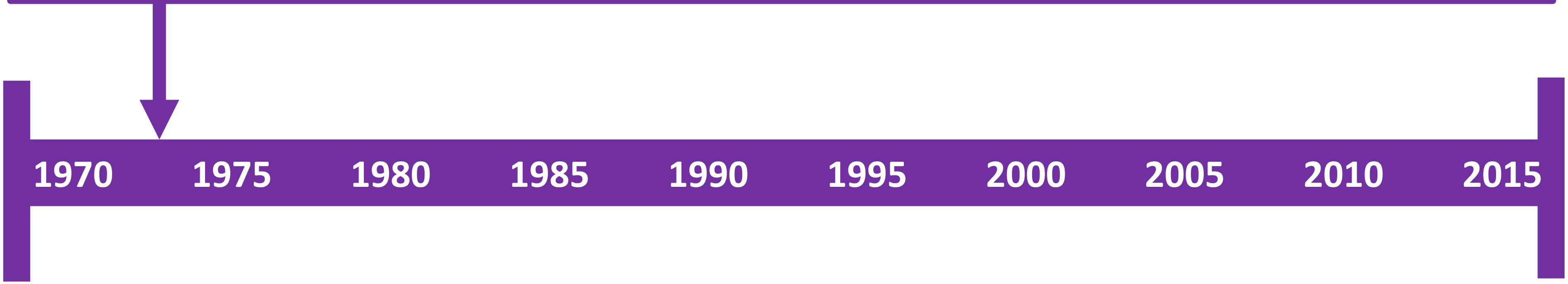
History of C

Timeline

1973

C

- As **Ritchie** worked on “New B”, it diverged more and more from the original B language and eventually the name was changed to C.
- By 1973, C was stable enough to start rewriting the UNIX operating system in. In addition to better supporting hardware like the PDP-11, C gave UNIX portability leading to increased interest in UNIX and its eventual fragmentation.



History of C

Timeline

1978-1980

The C Programming Language

- Work continued on C during the 1970s but a lack of official standardization or documentation quickly became an issue as other developers started creating their own C compilers.
- In 1978 the first book on C, “***The C Programming Language”*** (also referred to as K&R or the “White Book”) was published by **Brian Kernighan** and **Dennis Ritchie**.
- In the absence of official standards, K&R became the de facto C reference.

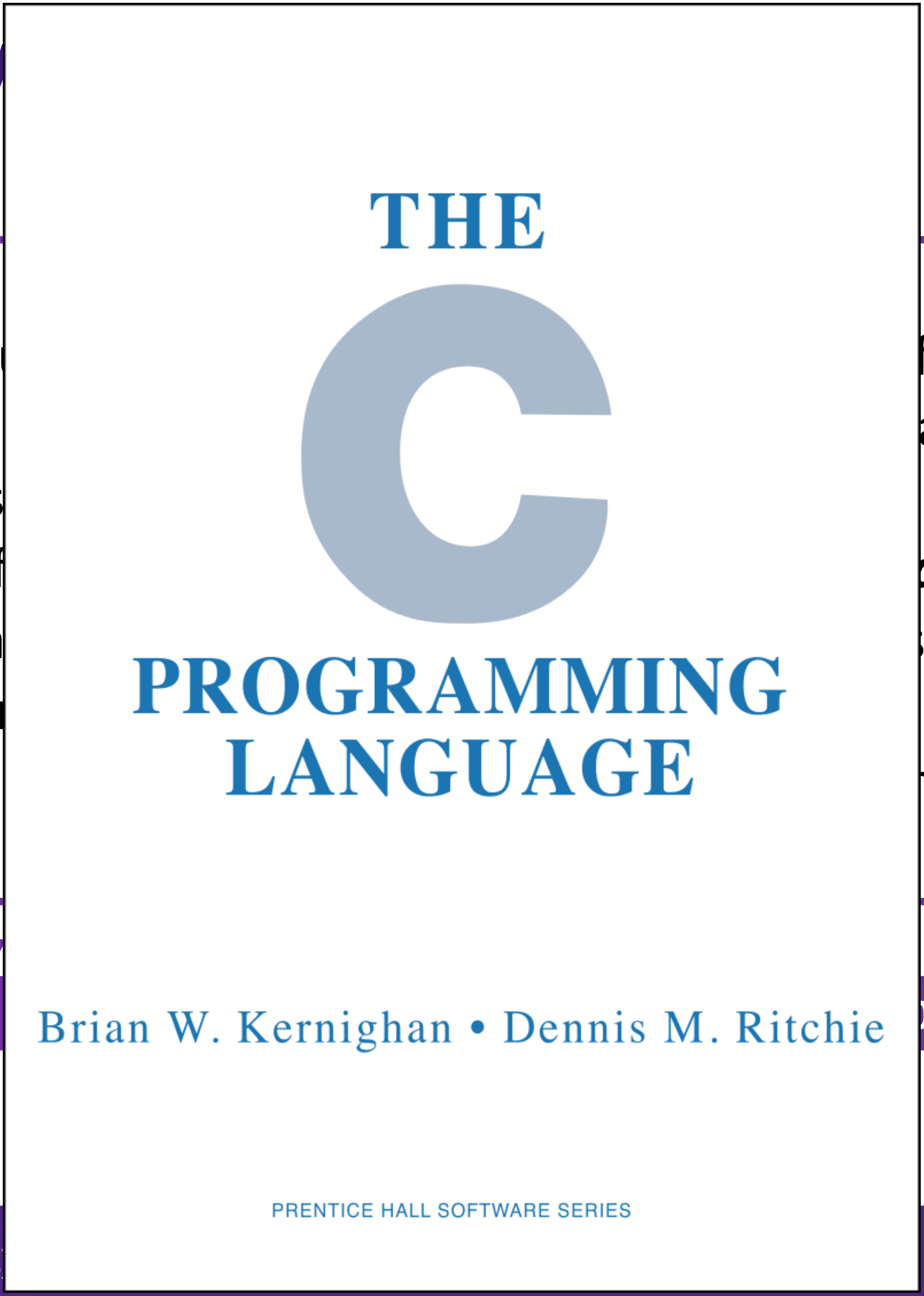
1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

History

Timeline

1978-1980

- Work contin f official standardizat an issue as other developers s
- In 1978 the f *anguage"* (also referred to a shed by **Brian Kernighan** a
- In the absen he de facto C reference.



THE

C

PROGRAMMING LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

PRENTICE HALL SOFTWARE SERIES

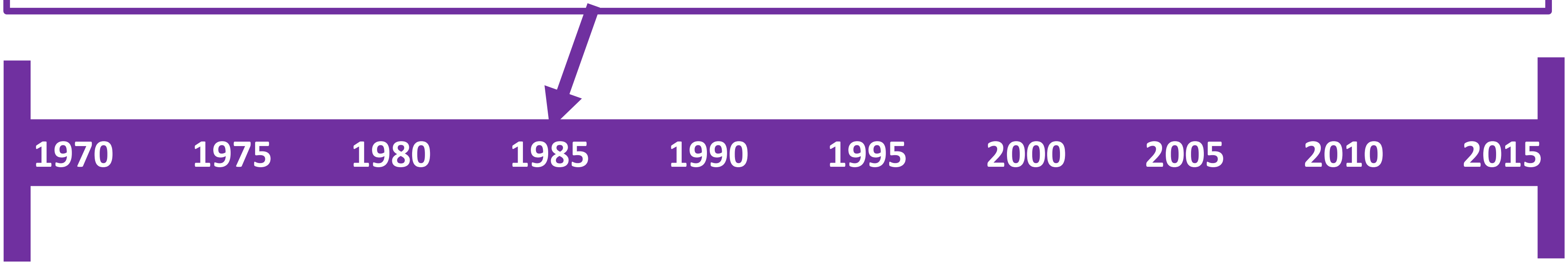
1970 1975 005 2010 2015

History of C
Timeline

1983-1989

ANSI C

- While K&R was a useful reference on the C programming language it was fuzzy or ambiguous about certain features. This complicated the creation of C compilers for new hardware.
- To remedy this situation, the American National Standards Institute (ASNI) began the standardization of C in 1983, producing a finalized standard in 1989 that is now often referred to as **C89** or **C90** (as opposed to **K&R C**).



1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

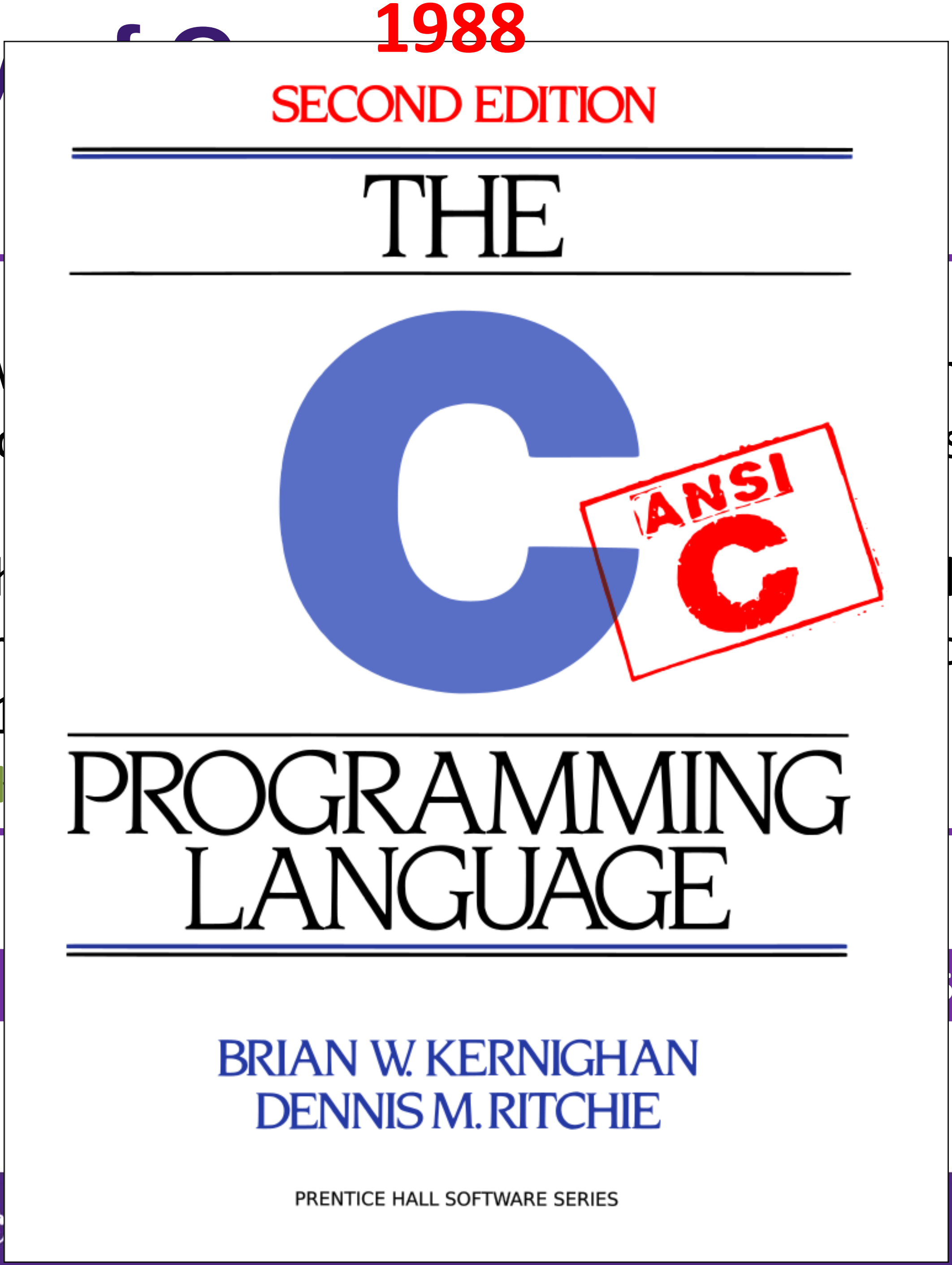
History

Timeline

1983-1989

- While K&R w ing language it was fuzzy (s complicated the creation
- To remedy th lards Institute (ASNI) begar cing a finalized standard in 1 or **C90** (as opposed to

1988



SECOND EDITION

THE C PROGRAMMING LANGUAGE

ANSI C

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES

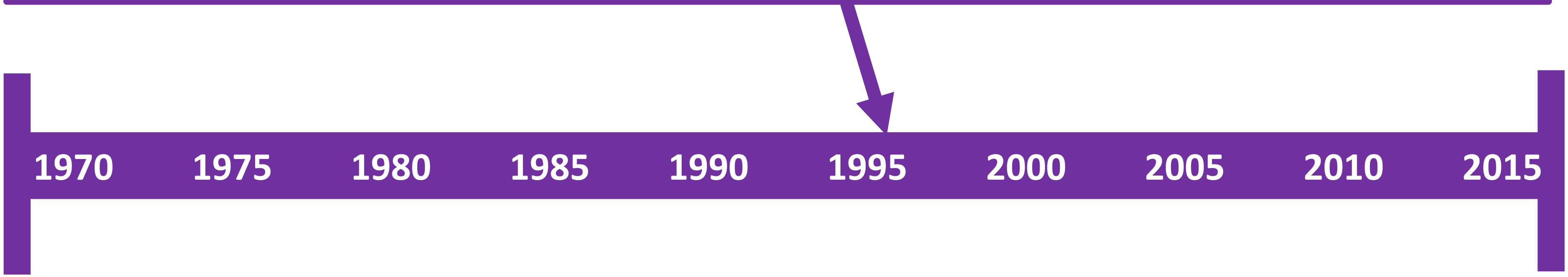
1970 1975 2010 2015

History of C
Timeline

1995-1999

C99

- In 1995 the C standard underwent further changes to modernize it and add things like one-line comments, new data types, and better floating point support.
- Changes were published as ISO/IEC 9899:1999 in 1999.
- This version of C is now referred to as **C99** and terms like ANSI C are less common as they are now ambiguous.

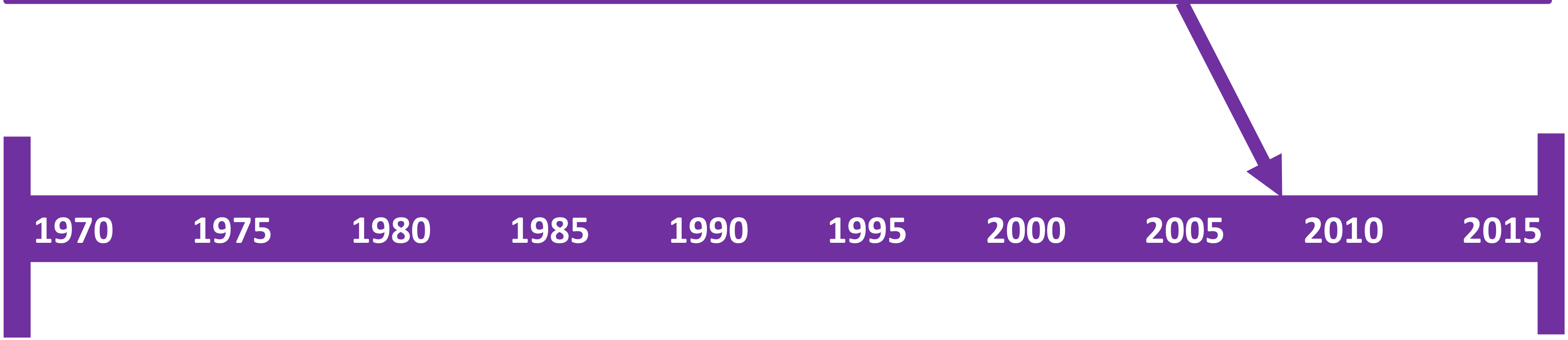


History of C
Timeline

2007-2011

C11

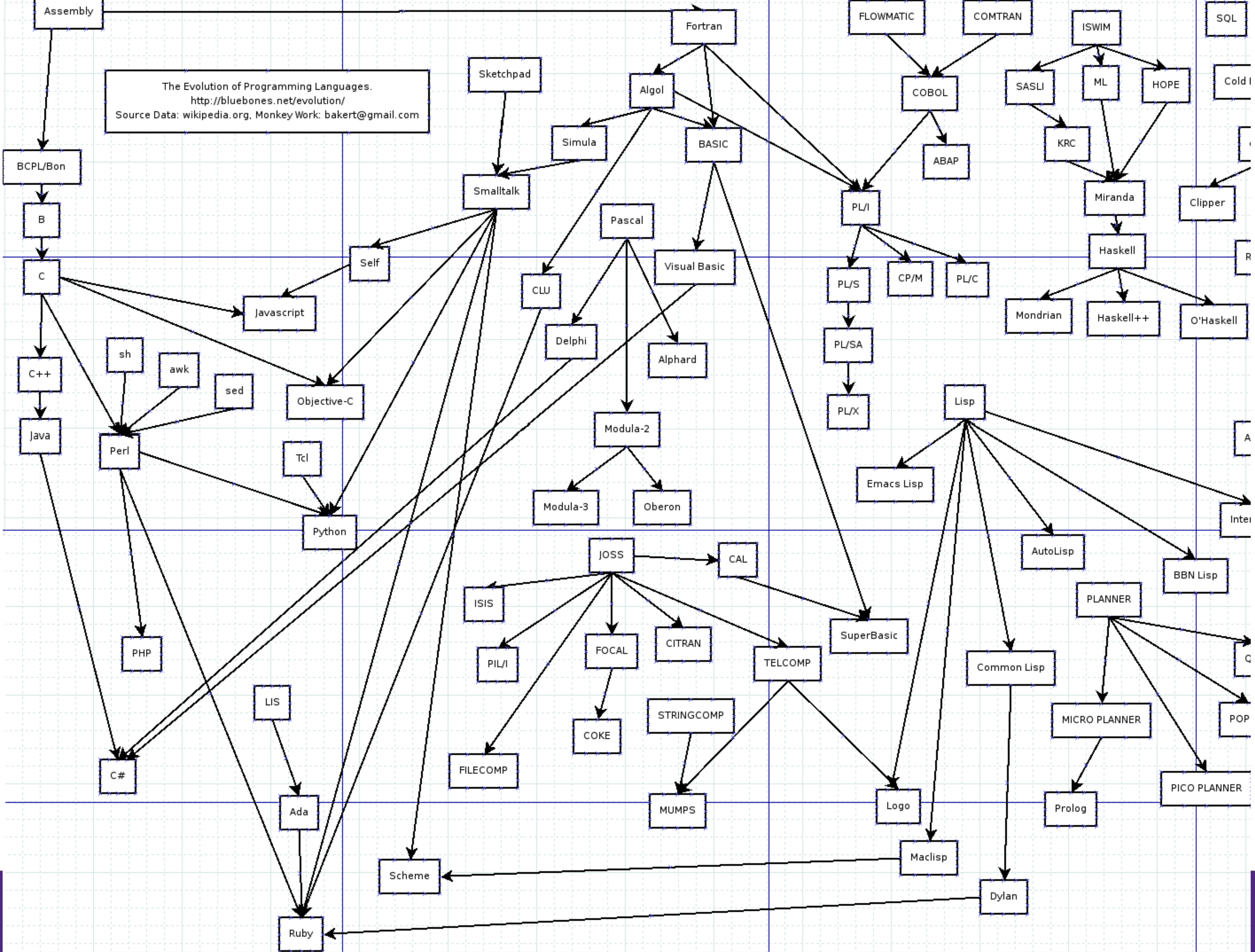
- In 2007 work started yet again to modernize and update the C standard.
- Adds things like improved Unicode support, multi-threading, removes the gets function (deprecated in C99), and more.
- Published in April 2011, and referred to as **C11**.



1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

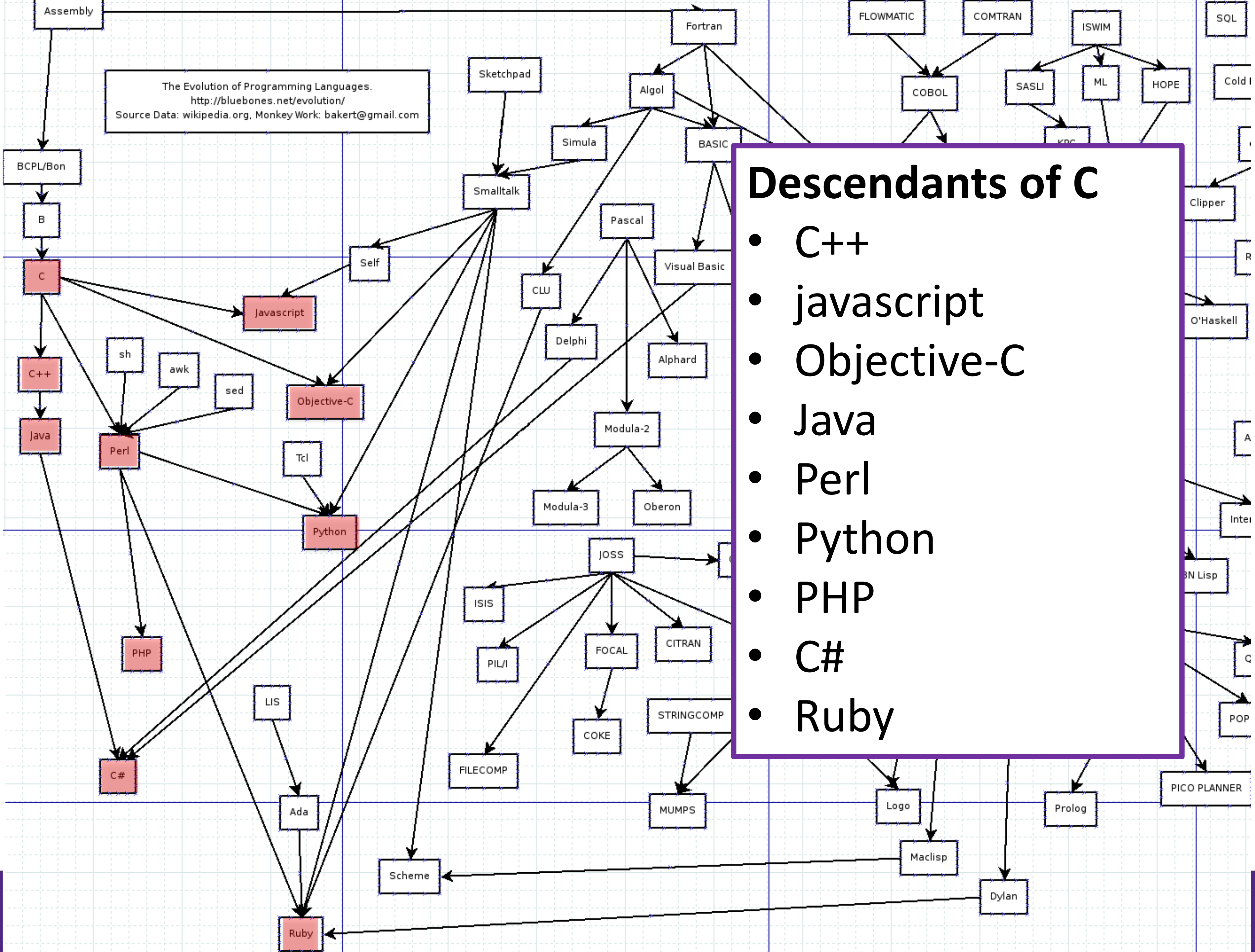
Relation to Other Languages

- Many modern programming languages borrowed heavily from the C syntax and feature set.
- **Examples:**
  - C++
  - Java
  - C#
  - Perl
- If you have used these in the past, the C syntax may be familiar to you.



The Evolution of Programming Languages.
http://bluebones.net/evolution/
Source Data: wikipedia.org, Monkey Work: bakert@gmail.com

Assembly
Fortran
FLOWMATIC
COMTRAN
ISWIM
SQL
Sketchpad
Algol
COBOL
SASLI
ML
HOPE
Cold I
BCPL/Bon
Simula
BASIC
ABAP
KRC
Smalltalk
PL/I
Miranda
Clipper
B
Pascal
Haskell
R
C
Self
Visual Basic
CLU
PL/S
CP/M
PL/C
Javascript
Delphi
Mondrian
Haskell++
O'Haskell
sh
awk
C++
PL/SA
sed
Alphard
Objective-C
Lisp
Java
PL/X
Perl
Tcl
Modula-2
A
Emacs Lisp
Modula-3
Oberon
Python
Inter
JOSS
CAL
AutoLisp
BBN Lisp
ISIS
PLANNER
SuperBasic
PHP
PIL/I
FOCAL
CITRAN
TELCOMP
Common Lisp
Q
LIS
STRINGCOMP
MICRO PLANNER
POP
COKE
C#
FILECOMP
PICO PLANNER
Ada
MUMPS
Logo
Prolog
Maclisp
Scheme
Dylan
Ruby


The Evolution of Programming Languages.
http://bluebones.net/evolution/
Source Data: wikipedia.org, Monkey Work: bakert@gmail.com

Descendants of C

- C++
- javascript
- Objective-C
- Java
- Perl
- Python
- PHP
- C#
- Ruby

Why C?

Philosophy

- Low-level
  - C allows us to study and access low level concepts that higher-level programming languages try to hide.
  - C is closer to assembly while still being easy to use and portable.
- Small Language
  - C is lightweight, containing a limited necessary set of features.
  - Useful for embedded systems or use cases where language overhead can be an issue.
  - *“C is not a big language, and it is not well served by a big book.”*
    — **Brian Kernighan**, The C Programming Language

Why C?

Philosophy

- Permissive
  - C does not hold your hand, assumes you know what you are doing.
  - Easy to make errors, but provides more latitude than you would find in other languages.
  - *“[C has] the power of assembly language and the convenience of … assembly language.”* — **Dennis Ritchie**

Why C?

Strengths

- Efficiency
  - C’s philosophy of being light weight and history of being used on systems with minimal resources has resulted in a language that is ideal when every bit of memory or CPU cycle is needed.
- Portability
  - C is portable in the sense that C programs can be compiled for a wide range of platforms.
  - Not necessarily the modern concept of portability as seen in Java, Python, etc.
- Integration with UNIX/Linux

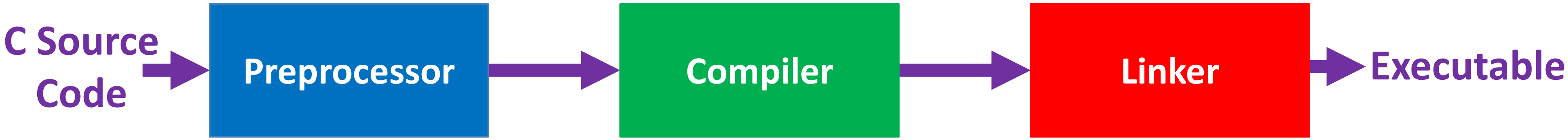
Why C?

Weaknesses

- Programs can be error-prone.
- Programs can be difficult to understand.
- Programs can be difficult to modify.

C Fundamentals

Compiling and Linking



- **Preprocessor**
  - Deals with ***directives***, commands that start with #.
  - Does textual edits and replacements to code (e.g. removes comments from code before compiling).
- **Compiler**
  - Translates source code into machine instructions (***object*** code).
- **Linker**
  - Combines the object code of our program with any additionally required ***object*** code (e.g. code from shared libraries).

Compiling and Linking

A Simple Example:

/cs2211/week6/hello.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

Compiling and Linking

A Simple Example: How to run

Compiling and Linking

A Simple Example: How to run

```
[dservos5@cs2211b week6]$ gcc -o hello hello.c

[dservos5@cs2211b week6]$ hello
Hello World!
```

Compiling and Linking

gcc

- GNU Compiler Collection (GCC)
- One of the most popular C compilers.
- Does preprocessing, compiling and linking for one in one command.
- UNIX command is `gcc`:

```
gcc -o EXECUTABLE_FILE_NAME C_FILE_NAME
```

Compiling and Linking

gcc

- GNU Compiler Collection (GCC)
- One of the most popular C compilers.
- Does preprocessing, compiling and linking for one in one command.
- UNIX command is gcc:

Name of executable that will be created

Name of your c source code file to compile

```
gcc -o EXECUTABLE_FILE_NAME C_FILE_NAME
```

Compiling and Linking

gcc

What about cc?

cc is normally the default compiler on modern UNIX like systems. On the course server it is a link to gcc:

```
[dservos5@cs2211b week6]$ ls -l /usr/bin/cc
lrwxrwxrwx. 1 root root 3 Jan  5 07:41 /usr/bin/cc -> gcc
```

- UNIX command is gcc:

Name of executable that will be created

Name of your c source code file to compile

```
gcc -o EXECUTABLE_FILE_NAME C_FILE_NAME
```

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```
#include <stdio.h>

int ma


}
```

# indicates that this is a directive dealt with by the preprocessor.

`#include` tells the preprocessor to essentially copy and paste this file into our code.

Used to tell C that we will be using functions from this library. In this case, this is the standard input/output library that contains functions like `printf` and `scanf` for printing output and reading in input.

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

Every C program needs a main function.

This is a special function that acts as the main entry point to your program (the first function that is run).

{ and } are used to specify what code is contained inside the function. In this case, the `printf` and `return` lines are inside of the main function.

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

int here describes what data type the function returns.

The return of the main function is always the programs exit status. You should always ensure that your main function returns a meaningful integer exit status (i.e. 0 for success, 1 to 255 for failure).

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

The return keyword specifies the value returned by the function and causes the function to exit at this point.

In the case of the main function, the returned value is the exit status.

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

A list of parameters a function takes may be given between the ()s. We will discuss this more when we talk about functions and arguments.

To denote that a function does not take any parameters, you can give no values between the ()s or do as follows:

```
int main(void) {
```

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

A list of parameters a function takes may be given between the ()s. We will discuss this more when we talk about functions and arguments.

To denote that a fur
rs you
can give no values b

This is technically more correct and considered better practice.

```
int main(void) {
```

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

The printf function is very similar to the printf command in UNIX.

It outputs a string to the standard output, replacing metacharacters like \n or \t with their corresponding special character.

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

Note that in C, every statement needs to be ended with a semicolon (;).

We can put multiple statements on one line using semicolons such as:

```c
printf("Hello World!\n"); return 0;
```

but this is considered bad practice in most cases (makes code harder to read).

Compiling and Linking

A Simple Example: Closer Look

/cs2211/week6/hello.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

This does not apply to preprocessor directives like #include or the opening/closing of blocks of code (e.g. with {}s).

Compiling and Linking

A Simple Example: Update the code to add a line that says "Good Bye" before exiting.

/cs2211/week6/hello.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        return 0;
}
```

Compiling and Linking

A Simple Example: Update the code to add a line that says “Good Bye” before exiting.

/cs2211/week6/hellob.c

```c
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        printf("Good Bye\n");
        return 0;
}
```

Compiling and Linking

A Simple Example: Update the code to add a line that says “Good Bye” before exiting.

/cs2211/week6/hellob.c

```
#include <stdio.h>

int main() {
        printf("Hello World!\n");
        printf("Good Bye\n");
        return 0;
}
```

Need to recompile before running again!

General Form

The general form of most simple C programs is:

```
directives
int main(void) {
   declarations
   statements
   return 0;
}
```

General Form

The general form of most simple C programs is:

```
directives
int main(void) {
   declarations
   statements
   return 0;
}
```

Directives

- Directives to the preprocessor to include files, define constants, etc.
- If we use a built in function from a library, we need to include it here. For example, `printf` is from the stdlib library, so we need `#include <stdlib.h>` here if we want to use `printf`.

General Form

The general form of most simple C programs is:

```
directives
int main(void) {
   declarations
   statements
   return 0;
}
```

Declarations of Variables

- Variables in C must be declared before used in statements or assigned values.
- In older C standards, this had to be done before statements. In modern standards so long as the variable is declared before it is used, the order does not matter.

General Form

The general form of most simple C programs is:

```
directives
int main(void) {
   declarations
   statements
   return 0;
}
```

Statements

- Statements like calling a function (e.g. `printf`), variable assignment, etc.

Comments

- Comments are lines that are removed by the preprocessor and not interpreted as code to be compiled.
- Similar to # in shell scripts.
- C99 supports both one line and multiline comments. Older C standards only supported multiline comments.

```c
/* This is an example of
   a multiline comment that
   can span multiple lines */

// This is an example of a one line comment

printf("Hi"); // It can be put after a command.
```

Variables and Data Types

- C is a strongly typed language.
- All variables need to be declared with a data type that specifies the type of data they can hold.
- **Supported Primitive Integer Data Types:**

Name	Minimum Range	Size (on course server)
short	−32,767 to 32,767	2 Bytes
unsigned short	0 to 65535	2 Bytes
int	−32,767 to 32,767	4 Bytes
unsigned int	0 to 65535	4 Bytes
long	−2,147,483,647 to 2,147,483,647	8 Bytes
unsigned long	0 to 4,294,967,295	8 Bytes
long long	−9,223,372,036,854,775,807 to 9,223,372,036,854,775,807	8 Bytes
unsigned long long	0 to 18,446,744,073,709,551,615	8 Bytes

Varia

- C is a s
- All var es
  the ty
- **Suppo**

Integer types are defined by the minimum range they can hold.

Can be of any size so long as they can hold this range.

Can be different machine to machine and compiler to compiler.

Name	Minimum Range	Size (on course server)
short	−32,767 to 32,767	2 Bytes
unsigned short	0 to 65535	2 Bytes
int	−32,767 to 32,767	4 Bytes
unsigned int	0 to 65535	4 Bytes
long	−2,147,483,647 to 2,147,483,647	8 Bytes
unsigned long	0 to 4,294,967,295	8 Bytes
long long	−9,223,372,036,854,775,807 to 9,223,372,036,854,775,807	8 Bytes
unsigned long long	0 to 18,446,744,073,709,551,615	8 Bytes

Varia

We can find the size of a type using the `sizeof` function in stdlib.

More on this in another lecture but you can try out the code in */cs2211/week6/csize.c* to print the sizes of each type.

- C is a
- All va es the ty
- **Supp**

Name	Minimum Range	Size (on course server)
short	−32,767 to 32,767	2 Bytes
unsigned short	0 to 65535	2 Bytes
int	−32,767 to 32,767	4 Bytes
unsigned int	0 to 65535	4 Bytes
long	−2,147,483,647 to 2,147,483,647	8 Bytes
unsigned long	0 to 4,294,967,295	8 Bytes
long long	−9,223,372,036,854,775,807 to 9,223,372,036,854,775,807	8 Bytes
unsigned long long	0 to 18,446,744,073,709,551,615	8 Bytes

Variables and Data Types

- C is a strongly typed language.
- All variables need to be declared with a data type that specifies the type of data they can hold.
- **Supported Primitive Decimal Data Types:**

Name	Range (on course server)	Size (on course server)
float	-3.4e38 to 3.4e38	4 Bytes
double	-1.7e308 to 1.7e308	8 Bytes
long double	-3.362103e4932 to 1.189731e4932	16 Bytes

Not defined by minimum range.

Variables and Data Types

- C is a strongly typed language.
- All variables need to be declared with a data type that specifies the type of data they can hold.
- **Supported Primitive Character Data Types:**

Name	Minimum Range	Size (on course server)
char	Either −127 to 127 OR 0 to 255	1 Byte
signed char	−127 to 127	1 Byte
unsigned char	0 to 255	1 Byte

Technically these are Integer Data Types

Variables and Data Types

- C is a strongly typed language.
- All variables need to be declared with a data type that specifies the type of data they
- **Supported Primitive (**

Machine dependent. On the course server char is signed by default.

Name	Minimum Range	Size (on course server)
char	Either −127 to 127 OR 0 to 255	1 Byte
signed char	−127 to 127	1 Byte
unsigned char	0 to 255	1 Byte

Technically these are Integer Data Types

Variables and Data Types

- Variables names (identifiers) have to follow rules (just like shell variables).
- In C an identifier may contain letters, digits, and underscore.
- Must start with a letter or underscore.
- Can not be a reserved keyword (e.g. int, void, return, etc.).
- Identifiers are case-sensitive (capitalization matters).
- **Which of the following are valid?**

times10
10times
get_next_char
get-next-char
return

1stplace
firstplace
_height
___width
INT

myvoid
_123_
abc123
CAT
bAt_

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

```
int age;
float balance;
char  initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

Declares a single signed integer named age.

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

Declares a single float named balance.

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Declares a single char named initial.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Declares three signed integers named x, y and z.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Declares three doubles named a, b and c.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Declares a single signed integer named w and assigns it an initial value of -456.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **Examples:**

```
int age;
float balance;
char initial;

int x, y, z;
double a, b, c;

int w = -456;
unsigned int height = 123;
```

Declares a single unsigned integer named height and assigns it an initial value of 123.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **More Examples:**

```
int height = 5, length = 3, width = 10;

int x, y, z = 10;

float pi = 3.14159f
long big = 100000000l
double small = 0.000000001d
```

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **More Examples:**

```
int height = 5, length = 3, width = 10;

int x, y, z = 10

float pi = 3.14159f
long big = 100000000l
double small = 0.000000001d
```

Declares three signed integers named height, length and width and gives them the values 5, 3 and 10 respectively.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **More Examples:**

```
int height = 5, length = 3, width = 10;

int x, y, z = 10;

float pi = 3.14159f
long big = 1000
double small = (
```

Declares three signed integers named x, y and z. Only z is given a value, the others are uninitialized.

Variables and Data Types

- Variables are declared by giving the data type name followed by the variable name.
- **More Examples:**

In some cases, we need to tell C what type a constant is. We do this with letter suffixes. u is for unsigned, f is for float, l for long and d for double.

Not always needed.

```
int height
int x, y,
float pi = 3.14159f
long big = 100000000l
double small = 0.000000001d
```

Variables and Data Types

- We can assign variables values after they have been declared with the = operator.
- **Example:**

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

x = 1;
y = -10;
z = 567;
w = y;
y = 20;
```

Variables and Data Types

- We can assign variables values after they have been declared with the = operator.
- **Example:**

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

x = 1;
y = -10;
z = 567;
w = y;
y = 20;
```

Result after all lines are run:

Variable	Value
x	1
y	20
z	567
w	-10
y	20

Operators

- Unlike shell scripts, C has built in arithmetic expressions and evaluation.
- The following arithmetic operators are supported:

Operator	Description	Example
+	Adds two operands.	A + B
–	Subtracts second operand from the first.	A − B
*	Multiplies both operands.	A * B
/	Divides numerator by de-numerator.	B / A
%	Modulus Operator and remainder of after an integer division.	B % A
++	Increment operator increases the integer value by one.	A++
--	Decrement operator decreases the integer value by one.	A--

Operators

- The following comparison operators are supported:

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	A == B
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	A != B
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	A > B
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	A < B
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	A >= B
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	A <= B

Operators

- The following comparison operators are supported:

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	A == B
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition	A != B
>		A > B
<		A < B
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	A >= B
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	A <= B

True result equals: 1

False result equals: 0

No Boolean data type by default (in C99 and up, one can be included from a library)

Operators

- The following logical operators are supported:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	A && B
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	A || B
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B)

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

```
int x = 10, y = 2, a, b, c;
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

```
int x = 10, y = 2, a, b, c;
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Variable	Value
x	
y	
a	
b	
c	

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

x and y initialized to 10 and 2

```
int x = 10, y = 2, a, b, c;
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Variable	Value
x	10
y	2
a	
b	
c	

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

```
int x = 10, y
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Value of a is set to:

x + y

= 10 + 2

= 12

Variable	Value
x	10
y	2
a	12
b	
c	

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

```
int x = 10, y
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Value of b is set to:

x * y

= 10 * 2

= 20

Variable	Value
x	10
y	2
a	12
b	20
c	

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

Value of y is incremented by 1.

```
int x = 10, y
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Variable	Value
x	10
y	3
a	12
b	20
c	

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

Value of x is decremented by 1.

```
int x = 10, y = 2, a, b, c;
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Variable	Value
x	9
y	3
a	12
b	20
c	

Operators

- We can use arithmetic expression to make calculations and store the result in variables:
- **Examples**:

```
int x = 10, y
a = x + y;
b = x * y;
y++;
x--;
c =  (x + y) / 2;
```

Value of c is set to:
(x + y) / 2
= (9 + 3) / 2
= (12) / 2
= 12 / 2
= 6

Variable	Value
x	9
y	3
a	12
b	20
c	6