Geneza C++, manipulatory

Bogdan Kreczmer

bogdan.kreczmer@pwr.edu.pl

Katedra Cybernetyki i Robotyki
Politechnika Wrocławska

Kurs: Programowanie obiektowe

Niniejsza prezentacja została wykonana przy użyciu systemu składu LaTeX *oraz stylu* `beamer`, *którego autorem jest Till Tantau.*

Strona domowa projektu `Beamer`:
`http://latex-beamer.sourceforge.net`

Plan prezentacji

Wzorce projektowe – czym są

Wzorzec projektowy (ang. design pattern) jest to rozwiązanie, które często jest stosowane przy rozwiązywaniu pewnej grupy problemów. Jest ono więc dobrze sprawdzonym i ugruntowanym podejściem, do do rozwiązywania danej klasy problemów projektowych.

Wzorze projektowy należy traktować jako opis rozwiązania, a nie gotową implementację.

Fabryka obiektów

- Fabryka obiektów
- Singleton

Przykłady implementacji uproszczonej fabryki oraz singletona dostarczone zostały w materiałach do wykładu nr 10.

Bjarne Stroustrup

Bjarne Stroustrup — (ur. 30.12.1950 – Århus, Dania) jest twórcą języka C++ i nadal aktywnie uczestniczy w jego dalszym rozwoju. Inspiracją do stworzenia tego typu języka były problemy i spostrzeżenia, których dokonał w trakcie swoich prac nad doktoratem w Uniwersytecie Cambridge.

Plan prezentacji

Tło akcji

Bezpośrednim przyczynkiem do powstania koncepcji języka C++ były prace Stroustrupa w Cambridge nad doktoratem. W trakcie swoich prac miał zbadać różne organizacje systemowego oprogramowania w systemach rozproszonych.

Zdarzenia

- Udana (w sensie konstrukcji aplikacji) realizacja symulatora systemu rozproszenego z wykorzystaniem *Simuli*. Pojęcie klasy w *Simuli* było bardzo pomocne na etapie projektu i implmentacji.
- Poważne problemy związne z wydajnością programów napisanego w *Simuli* zmusza do poszukiwania bardziej wydajnego narzędzia.
- Traumatycznie bolesne doświadczenia związane z koniecznością implementacji symulatora w oparciu o język *BCPL*. Osiągnięta została bardzo dobra wydajność kosztem wielu problemów przy realizacji implementacji.

Zdarzenia

- Udana (w sensie konstrukcji aplikacji) realizacja symulatora systemu rozproszenego z wykorzystaniem *Simuli*. Pojęcie klasy w *Simuli* było bardzo pomocne na etapie projektu i implmentacji.
- Poważne problemy związne z wydajnością programów napisanego w *Simuli* zmusza do poszukiwania bardziej wydajnego narzędzia.
- Traumatycznie bolesne doświadczenia związane z koniecznością implementacji symulatora w oparciu o język *BCPL*. Osiągnięta została bardzo dobra wydajność kosztem wielu problemów przy realizacji implementacji.

Zdarzenia

- Udana (w sensie konstrukcji aplikacji) realizacja symulatora systemu rozproszenego z wykorzystaniem *Simuli*. Pojęcie klasy w *Simuli* było bardzo pomocne na etapie projektu i implmentacji.
- Poważne problemy związne z wydajnością programów napisanego w *Simuli* zmusza do poszukiwania bardziej wydajnego narzędzia.
- Traumatycznie bolesne doświadczenia związane z koniecznością implementacji symulatora w oparciu o język *BCPL*. Osiągnięta została bardzo dobra wydajność kosztem wielu problemów przy realizacji implementacji.

Zdarzenia

- Udana (w sensie konstrukcji aplikacji) realizacja symulatora systemu rozproszenego z wykorzystaniem *Simuli*. Pojęcie klasy w *Simuli* było bardzo pomocne na etapie projektu i implmentacji.
- Poważne problemy związne z wydajnością programów napisanego w *Simuli* zmusza do poszukiwania bardziej wydajnego narzędzia.
- Traumatycznie bolesne doświadczenia związane z koniecznością implementacji symulatora w oparciu o język *BCPL*. Osiągnięta została bardzo dobra wydajność kosztem wielu problemów przy realizacji implementacji.

Epilog

Po zakończeniu prac nad doktoratem Stroustrup postanawia stworzyć język programowania, który łączyłby koncepcje klas z *Simuli* oraz elastyczność tego języka z efektywnością kodu języka *BCPL*.

Plan prezentacji

Niektóre spostrzeżenia

- Kompilator *Simuli* dobrze wychwytywał błędy typu. Błędy te przeważnie były albo wynikiem "głupich" pomyłek w trakcie programowania albo też potknięć koncepcyjnych.

  Ten drugi rodzaj błędów ma wręcz fundamentalne znaczenie dla procesu projektowania i pisania programownia *(autor kursu pozwolił sobie tę uwagę wzmocnić).*

- Żaden inny prosty system sprawdzania ścisłej zgodności typów nie dostarczył tego rodzaju wsparcia.

  Przykładem może być system *Pascala*. Sprawiał on więcej kłopotów niż pozwalał ich uniknąć. Zmuszał do modyfikacji projektu, aby dostosować go do warunków narzuconych przez implementację *(co nie zmienia faktu, że jest to bardzo dobry język dla początkowego kursu nauki informatyki – uwaga autora kursu)*.

Niektóre spostrzeżenia

- Kompilator *Simuli* dobrze wychwytywał błędy typu. Błędy te przeważnie były albo wynikiem "głupich" pomyłek w trakcie programowania albo też potknięć koncepcyjnych.

  Ten drugi rodzaj błędów ma wręcz fundamentalne znaczenie dla procesu projektowania i pisania programownia *(autor kursu pozwolił sobie tę uwagę wzmocnić).*

- Żaden inny prosty system sprawdzania ścisłej zgodności typów nie dostarczył tego rodzaju wsparcia.

  Przykładem może być system *Pascala*. Sprawiał on więcej kłopotów niż pozwalał ich uniknąć. Zmuszał do modyfikacji projektu, aby dostosować go do warunków narzuconych przez implementację *(co nie zmienia faktu, że jest to bardzo dobry język dla początkowego kursu nauki informatyki – uwaga autora kursu)*.

Niektóre spostrzeżenia

- Kompilator *Simuli* dobrze wychwytywał błędy typu. Błędy te przeważnie były albo wynikiem "głupich" pomyłek w trakcie programowania albo też potknięć koncepcyjnych.

  Ten drugi rodzaj błędów ma wręcz fundamentalne znaczenie dla procesu projektowania i pisania programownia *(autor kursu pozwolił sobie tę uwagę wzmocnić).*

- Żaden inny prosty system sprawdzania ścisłej zgodności typów nie dostarczył tego rodzaju wsparcia.

  Przykładem może być system *Pascala*. Sprawiał on więcej kłopotów niż pozwalał ich uniknąć. Zmuszał do modyfikacji projektu, aby dostosować go do warunków narzuconych przez implementację *(co nie zmienia faktu, że jest to bardzo dobry język dla początkowego kursu nauki informatyki – uwaga autora kursu)*.

Niektóre spostrzeżenia

- Kompilator *Simuli* dobrze wychwytywał błędy typu. Błędy te przeważnie były albo wynikiem "głupich" pomyłek w trakcie programowania albo też potknięć koncepcyjnych.

  Ten drugi rodzaj błędów ma wręcz fundamentalne znaczenie dla procesu projektowania i pisania programownia *(autor kursu pozwolił sobie tę uwagę wzmocnić).*

- Żaden inny prosty system sprawdzania ścisłej zgodności typów nie dostarczył tego rodzaju wsparcia.

  Przykładem może być system *Pascala*. Sprawiał on więcej kłopotów niż pozwalał ich uniknąć. Zmuszał do modyfikacji projektu, aby dostosować go do warunków narzuconych przez implementację *(co nie zmienia faktu, że jest to bardzo dobry język dla początkowego kursu nauki informatyki – uwaga autora kursu)*.

Niektóre spostrzeżenia

- Pojęcie klasy było zasadniczym elementem różniącym „sztywny” *Pascal* od „elastycznej” *Simuli*.
- Wyczerpujące sprawdzanie zgodności typów w *Simuli* sprawiało, że liczba problemów i błędów nie wzrastała szybciej niż liniowo wraz ze wzrostem programu.

Niektóre spostrzeżenia

- Pojęcie klasy było zasadniczym elementem różniącym „sztywny” *Pascal* od „elastycznej” *Simuli*.
- Wyczerpujące sprawdzanie zgodności typów w *Simuli* sprawiało, że liczba problemów i błędów nie wzrastała szybciej niż liniowo wraz ze wzrostem programu.

Plan prezentacji

Ważniejsze daty

1979 – (*Maj*) Początek prac na *C z Klasami*. (*Październik*) Oddanie do użytku pierwszej implementacji tego języka.

1983 – (*Sierpień*) Oddanie do użytku pierwszej implementacji języka *C++*. (*Grudzień*) Powstanie nazwy *C++*.

1985 – (*Luty*) Powstanie pierwszej wersji języka *C++* (wersja E) do użytku zewnętrznego. (*Październik*) Ukazanie się pierwszego wydania książki "Język C++".

1987 – (*Grudzień*) Ukazanie się kompilatora *GNU C++*.

Ważniejsze daty

1979 – (*Maj*) Początek prac na *C z Klasami*. (*Październik*) Oddanie do użytku pierwszej implementacji tego języka.

1983 – (*Sierpień*) Oddanie do użytku pierwszej implementacji języka *C++*. (*Grudzień*) Powstanie nazwy *C++*.

1985 – (*Luty*) Powstanie pierwszej wersji języka *C++* (wersja E) do użytku zewnętrznego. (*Październik*) Ukazanie się pierwszego wydania książki "Język C++".

1987 – (*Grudzień*) Ukazanie się kompilatora *GNU C++*.

Ważniejsze daty

1979 – (*Maj*) Początek prac na *C z Klasami*. (*Październik*) Oddanie do użytku pierwszej implementacji tego języka.

1983 – (*Sierpień*) Oddanie do użytku pierwszej implementacji języka *C++*. (*Grudzień*) Powstanie nazwy *C++*.

1985 – (*Luty*) Powstanie pierwszej wersji języka *C++* (wersja E) do użytku zewnętrznego. (*Październik*) Ukazanie się pierwszego wydania książki "Język C++".

1987 – (*Grudzień*) Ukazanie się kompilatora *GNU C++*.

Ważniejsze daty

1979 – (*Maj*) Początek prac na *C z Klasami*. (*Październik*) Oddanie do użytku pierwszej implementacji tego języka.

1983 – (*Sierpień*) Oddanie do użytku pierwszej implementacji języka *C++*. (*Grudzień*) Powstanie nazwy *C++*.

1985 – (*Luty*) Powstanie pierwszej wersji języka *C++* (wersja E) do użytku zewnętrznego. (*Październik*) Ukazanie się pierwszego wydania książki "Język C++".

1987 – (*Grudzień*) Ukazanie się kompilatora *GNU C++*.

Ważniejsze daty

1979 – (*Maj*) Początek prac na *C z Klasami*. (*Październik*) Oddanie do użytku pierwszej implementacji tego języka.

1983 – (*Sierpień*) Oddanie do użytku pierwszej implementacji języka *C++*. (*Grudzień*) Powstanie nazwy *C++*.

1985 – (*Luty*) Powstanie pierwszej wersji języka *C++* (wersja E) do użytku zewnętrznego. (*Październik*) Ukazanie się pierwszego wydania książki "Język C++".

1987 – (*Grudzień*) Ukazanie się kompilatora *GNU C++*.

Ważniejsze daty

1989 – (*Grudzień*) Powstanie zespołu ANSI X3J16 do spraw normalizacji języka *C++*.

1990 – (*Maj*) Powstawnie w firmie *Borland* pierwszej implementacji języka *C++*.
Lipiec: Przyjęcie koncepcji szablonów.
Listopad: Przyjęcie koncepcji wyjątków.

1990 – Wydanie książki: Ellis Margaret A., Stroustrup B.
„*The Annotated C++ Reference Manual*”,
Reading, MA, Addison-Wesley 1990.
Stał się on nieformalnym standardem języka, określanego jako *C++ ARM*.

Ważniejsze daty

1989 – (*Grudzień*) Powstanie zespołu ANSI X3J16 do spraw normalizacji języka *C++*.

1990 – (*Maj*) Powstawnie w firmie *Borland* pierwszej implementacji języka *C++*.
Lipiec: Przyjęcie koncepcji szablonów.
Listopad: Przyjęcie koncepcji wyjątków.

1990 – Wydanie książki: Ellis Margaret A., Stroustrup B. „*The Annotated C++ Reference Manual*", Reading, MA, Addison-Wesley 1990.
Stał się on nieformalnym standardem języka, określanego jako *C++ ARM*.

Ważniejsze daty

1989 – (*Grudzień*) Powstanie zespołu ANSI X3J16 do spraw normalizacji języka *C++*.

1990 – (*Maj*) Powstawnie w firmie *Borland* pierwszej implementacji języka *C++*.
Lipiec: Przyjęcie koncepcji szablonów.
Listopad: Przyjęcie koncepcji wyjątków.

1990 – Wydanie książki: Ellis Margaret A., Stroustrup B. „*The Annotated C++ Reference Manual*", Reading, MA, Addison-Wesley 1990.
Stał się on nieformalnym standardem języka, określanego jako *C++ ARM*.

Ważniejsze daty

1992 – (*Luty*) Powstanie pierwszej implementacji języka *C++* (zawierającej szablony i wyjątki) w firmie DEC. (*Marzec*) Powstanie pierwszej implementacji języka *C++* w firmie Microsoft. (*Maj*) Powstanie pierwszej implementacji języka *C++* w firmie IBM.

1993 – *Marzec*: Przyjęcie koncepcji identyfikowania typu podczas wykonywania programu.
Lipiec: Przyjęcie koncepcji przestrzeni nazw.

1998 – Przyjęcie standardu ANSI/ISO języka *C++*.

Ważniejsze daty

1992 – (*Luty*) Powstanie pierwszej implementacji języka *C++* (zawierającej szablony i wyjątki) w firmie DEC. (*Marzec*) Powstanie pierwszej implementacji języka *C++* w firmie Microsoft. (*Maj*) Powstanie pierwszej implementacji języka *C++* w firmie IBM.

1993 – *Marzec*: Przyjęcie koncepcji identyfikowania typu podczas wykonywania programu.
Lipiec: Przyjęcie koncepcji przestrzeni nazw.

1998 – Przyjęcie standardu ANSI/ISO języka *C++*.

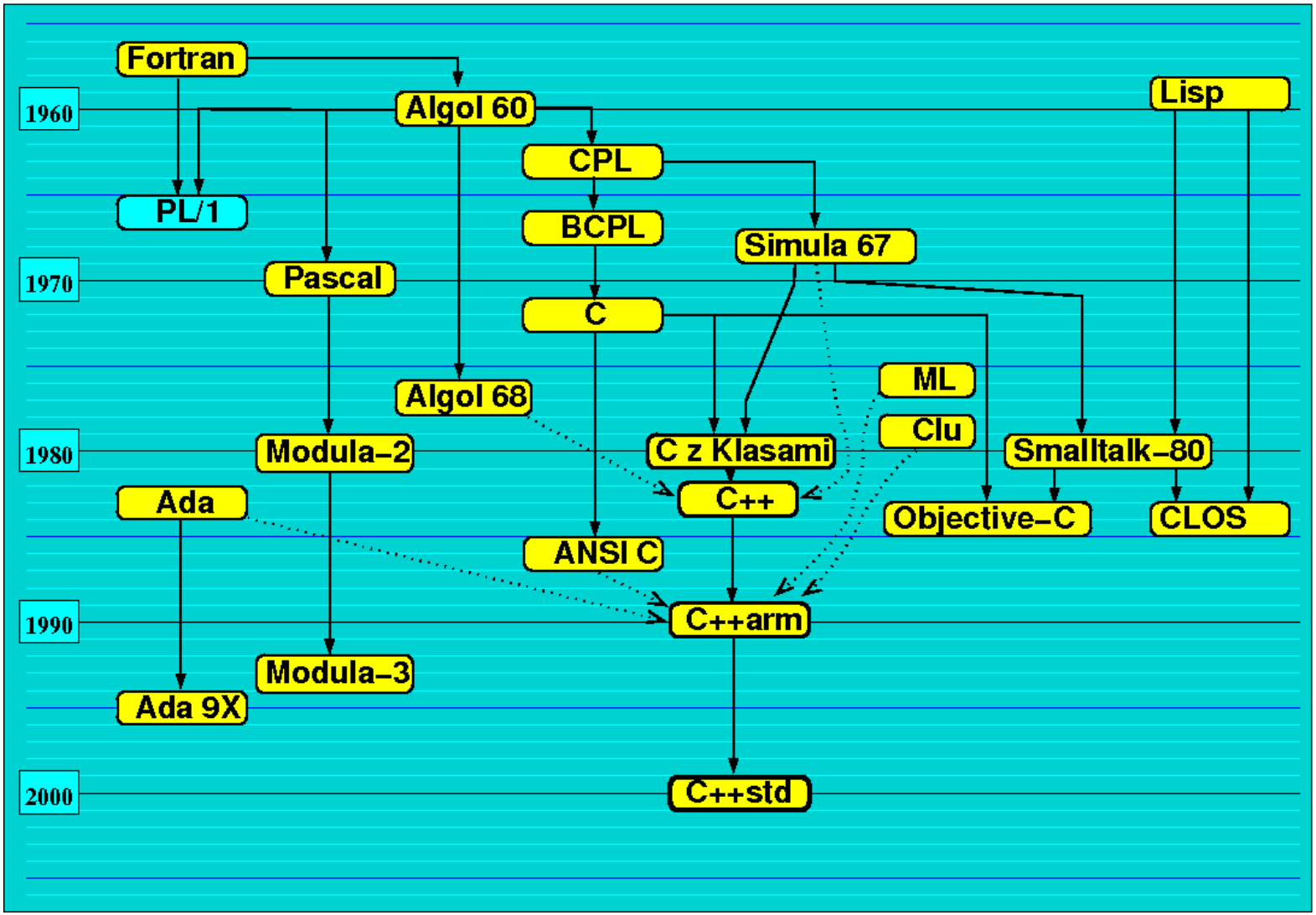
Ważniejsze daty

1992 – (*Luty*) Powstanie pierwszej implementacji języka *C++* (zawierającej szablony i wyjątki) w firmie DEC. (*Marzec*) Powstanie pierwszej implementacji języka *C++* w firmie Microsoft. (*Maj*) Powstanie pierwszej implementacji języka *C++* w firmie IBM.

1993 – *Marzec*: Przyjęcie koncepcji identyfikowania typu podczas wykonywania programu.
Lipiec: Przyjęcie koncepcji przestrzeni nazw.

1998 – Przyjęcie standardu ANSI/ISO języka *C++*.

Genealogia


1960
1970
1980
1990
2000
Fortran
Algol 60
Lisp
CPL
PL/1
BCPL
Simula 67
Pascal
C
Algol 68
ML
Clu
Modula–2
C z Klasami
Smalltalk–80
Ada
C++
Objective–C
CLOS
ANSI C
C++arm
Modula–3
Ada 9X
C++std

Dalszy rozwój i wprowadzane standardy

1998 – (ISO/IEC 14882:2011) Przyjęcie standardu ANSI/ISO języka *C++*.

2003 – (ISO/IEC 14882:2003) Korekcja wcześniejszego standardu.

2007 – (ISO/IEC TR 19768:2007) Oparty na technicznym raporcie *„Library Technical Report 1 ”*, który wprowadzał rozszerzenia do biblioteki standardowej.

2011 – (ISO/IEC 14882:2011) *11 sierpnia 2011*: Nowy standard języka *C++* wcześniej roboczo określany jako *C++0x*.

Dalszy rozwój i wprowadzane standardy

1998 – (ISO/IEC 14882:2011) Przyjęcie standardu ANSI/ISO języka *C++*.

2003 – (ISO/IEC 14882:2003) Korekcja wcześniejszego standardu.

2007 – (ISO/IEC TR 19768:2007) Oparty na technicznym raporcie *„Library Technical Report 1 ”*, który wprowadzał rozszerzenia do biblioteki standardowej.

2011 – (ISO/IEC 14882:2011) *11 sierpnia 2011*: Nowy standard języka *C++* wcześniej roboczo określany jako *C++0x*.

Dalszy rozwój i wprowadzane standardy

1998 – (ISO/IEC 14882:2011) Przyjęcie standardu ANSI/ISO języka *C++*.

2003 – (ISO/IEC 14882:2003) Korekcja wcześniejszego standardu.

2007 – (ISO/IEC TR 19768:2007) Oparty na technicznym raporcie *„Library Technical Report 1 ”*, który wprowadzał rozszerzenia do biblioteki standardowej.

2011 – (ISO/IEC 14882:2011) *11 sierpnia 2011*: Nowy standard języka *C++* wcześniej roboczo określany jako *C++0x*.

Dalszy rozwój i wprowadzane standardy

1998 – (ISO/IEC 14882:2011) Przyjęcie standardu ANSI/ISO języka *C++*.

2003 – (ISO/IEC 14882:2003) Korekcja wcześniejszego standardu.

2007 – (ISO/IEC TR 19768:2007) Oparty na technicznym raporcie *„Library Technical Report 1 ”*, który wprowadzał rozszerzenia do biblioteki standardowej.

2011 – (ISO/IEC 14882:2011) *11 sierpnia 2011*: Nowy standard języka *C++* wcześniej roboczo określany jako *C++0x*.

Dalszy rozwój i wprowadzane standardy

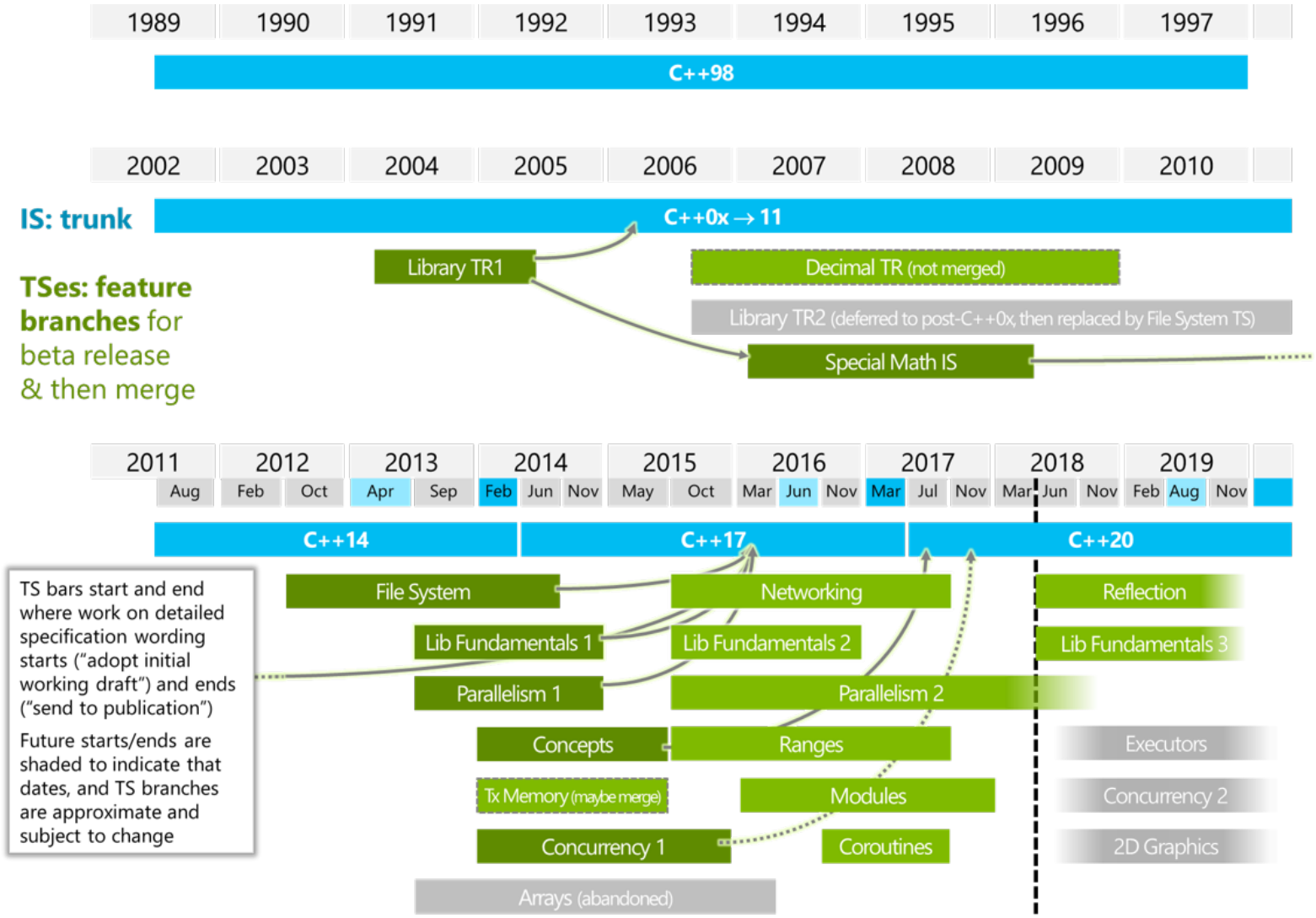
C++98 1998 – (ISO/IEC 14882:2011) Przyjęcie standardu ANSI/ISO języka *C++*.

C++03 2003 – (ISO/IEC 14882:2003) Korekcja wcześniejszego standardu.

C++TR1 2007 – (ISO/IEC TR 19768:2007) Oparty na technicznym raporcie *„Library Technical Report 1 ”*, który wprowadzał rozszerzenia do biblioteki standardowej.

C++11 2011 – (ISO/IEC 14882:2011) *11 sierpnia 2011*: Nowy standard języka *C++* wcześniej roboczo określany jako *C++0x*.

Aktualna mapa drogowa



Wzajemne powiązania

Kilka faktów:

- Słowo kluczowe **class** pochodzi z *Simuli* (podobnie jak notacja wskaźnika **this**).
- Zapis `funkcja(`**void**`)` w *ANSI C* pochodzi z języka *C z Klasami*.

Wzajemne powiązania

Kilka faktów:

- Słowo kluczowe **class** pochodzi z *Simuli* (podobnie jak notacja wskaźnika **this**).
- Zapis `funkcja(`**void**`)` w *ANSI C* pochodzi z języka *C z Klasami*.

Wzajemne powiązania

Kilka faktów:

- Słowo kluczowe **class** pochodzi z *Simuli* (podobnie jak notacja wskaźnika **this**).
- Zapis `funkcja(void)` w *ANSI C* pochodzi z języka *C z Klasami*.

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym* – nie ma wewnętrznych ograniczeń wykluczających możliwość napisania jakiegoś rodzaju programu, tzn. można korzystać w nim z większości technik programowania.
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym* – semantyka języka znajduje się na "niskim poziomie", tzn. podstawowe pojęcia języka odzwierciedlają podstawowe pojęcia tradycyjnego komputera.
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym* – implementacje tego języka istnieją zarówno dla mikrokomputerów jak też dla dużych superkomputerów.
- *C* jest językiem *przenośnym*

Dlaczego C

- *C* jest językiem *elastycznym*
- *C* jest językiem *wydajnym*
- *C* jest językiem *szeroko dostępnym*
- *C* jest językiem *przenośnym* – programy napisane w *C* zazwyczaj nie można automatycznie przenosić z jednego systemu operacyjnego do drugiego. Jednak przenośność z uwzględnieniem odpowiednich uwarunkowań jest możliwa.

Dlaczego C

Język *C*	Język *C++*
char Znak; scanf("%c", &Znak);	**int** Znak; cin >> Znak;

char Tab[10000]; scanf("%s", Tab);	**char** Tab[10000]; cin >> Tab;

Dlaczego C

Język *C*	Język *C++*
char Znak; scanf("%c", &Znak);	**int** Znak; cin >> Znak;

char Tab[10000]; scanf("%s", Tab);	**char** Tab[10000]; cin >> Tab;

Tak można, ale mimo to tak nie należy robić !!! (*niebezpieczeństwo niekontrolowanego przepełnienia)*

Formatowanie strumienia wejściowego

Język *C*	Język *C++*
char Tab[10]; scanf("%10s", Tab);	**char** Tab[10]; cin >> setw(sizeof(Tab)) >> Tab;

Formatowanie strumienia wejściowego

Język *C*	Język *C++*
char Tab[10]; scanf("%10s", Tab);	**char** Tab[10]; cin >> setw(sizeof(Tab)) >> Tab;
char Znak, Tab[10]; scanf("%c%10s", &Znak, Tab);	**char** Znak, Tab[10]; cin >> Znak >> setw(sizeof(Tab)) >> Tab;

Formatowanie strumienia wejściowego

Język *C*	Język *C++*
Wczytywanie bez pomijania znaków *białych*	
int KodZnaku; scanf("%c", &Znak);	**char** Znak; cin >> noskipws >> Znak;

Formatowanie strumienia wejściowego

Język *C*	Język *C++*
Wczytywanie bez pomijania znaków *białych*	
int KodZnaku; scanf("%c", &Znak);	**char** Znak; cin >> noskipws >> Znak;
Wymuszenie pomijania znaków *białych*	
char Znak; scanf(" %c", &Znak);	**char** Znak; cin >> skipws >> Znak;

Formatowanie strumienia wejściowego

Język *C*	Język *C++*
Wczytywanie bez pomijania znaków *białych*	
int KodZnaku; scanf("%c", &Znak);	**char** Znak; cin >> noskipws >> Znak;
Wymuszenie pomijania znaków *białych*	
char Znak; scanf(" %c", &Znak);	**char** Znak; cin >> skipws >> Znak;
Odrzucenie znaku	

int KodZnaku;
KodZnaku = getchar( );
ungetc(KodZnaku, stdin);
	char Znak;
	cin.get(Znak);
	cin.unget( );

Zestawienie dostępnych strumieni

Język *C*	Język *C++*
fprintf(stderr,"Komunikat o błędzie ...");	cerr << "Komunikat o błędzie ...";

Zestawienie dostępnych strumieni

Język *C*	Język *C++*
fprintf(stderr,"Komunikat o błędzie ...");	cerr << "Komunikat o błędzie ...";

Predefiniowane strumienie wejścia/wyjścia	
FILE* stdin	**istream** cin
FILE* stdout	**ostream** cout
FILE* stderr	**ostream** cerr
	ostream clog

Zestawienie dostępnych strumieni

Język *C*	Język *C++*
fprintf(stderr,"Komunikat o błędzie ...");	cerr << "Komunikat o błędzie ...";

Predefiniowane strumienie wejścia/wyjścia	
FILE* stdin	**istream** cin
FILE* stdout	**ostream** cout
FILE* stderr	**ostream** cerr
	ostream clog

Deklaracje plików nagłówkowych dla operacji wejścia/wyjścia	
#*include* <stdio.h>	#*include* <iostream> #*include* <iomanip> **using namespace** std;

Koniec prezentacji
Dziękuję za uwagę