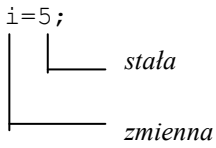
2. Typy danych

2.1. Zmienne i stałe

- Dane programu zapisywane są w postaci *zmiennych* lub *stałych*.
- *Zmienna* (ang. *variable*) to pewien obszar pamięci o nadanej symbolicznej nazwie, w którym można przechowywać dane, pobierać je i zmieniać podczas wykonywania programu.
- *Stała* (*literał*, ang. *literal constant*) to dana zapisana w sposób dosłowny (liczba, znak, napis), nie zmienia się podczas wykonywania programu.
- Przykład:



2.2. Identyfikatory czyli nazwy

- *Identyfikator* (ang. *identifier*) to nazwa służąca do oznaczania obiektów zdefiniowanych przez użytkownika w programie (zmiennych, funkcji, nazwanych stałych). Nazwa może zawierać litery, cyfry, znak podkreślenia, przy czym może rozpoczynać się *tylko od litery lub znaku podkreślenia*.
- Rozróżniane są małe i wielkie litery. Oznacza to, że `x` oraz `X` to dwie różne zmienne.
- Nazwa nie może być słowem kluczowym języka C++ (np. nazwą instrukcji `if`) ani być używana przez bibliotekę standardową.
- Standard ANSI C ogranicza rozmiar nazw do 31 znaków. Standard C++ ANSI/ISO nie nakłada ograniczeń na długość nazw. Jednakże większość kompilatorów przyjmuje swoje maksymalne długości nazw określane w opcjach kompilatora.
- Przykłady:

Poprawne nazwy	Niepoprawne nazwy
`licznik_12`	`12_licznik`
`test12`	`test-12`
`test_12`	`test..12`

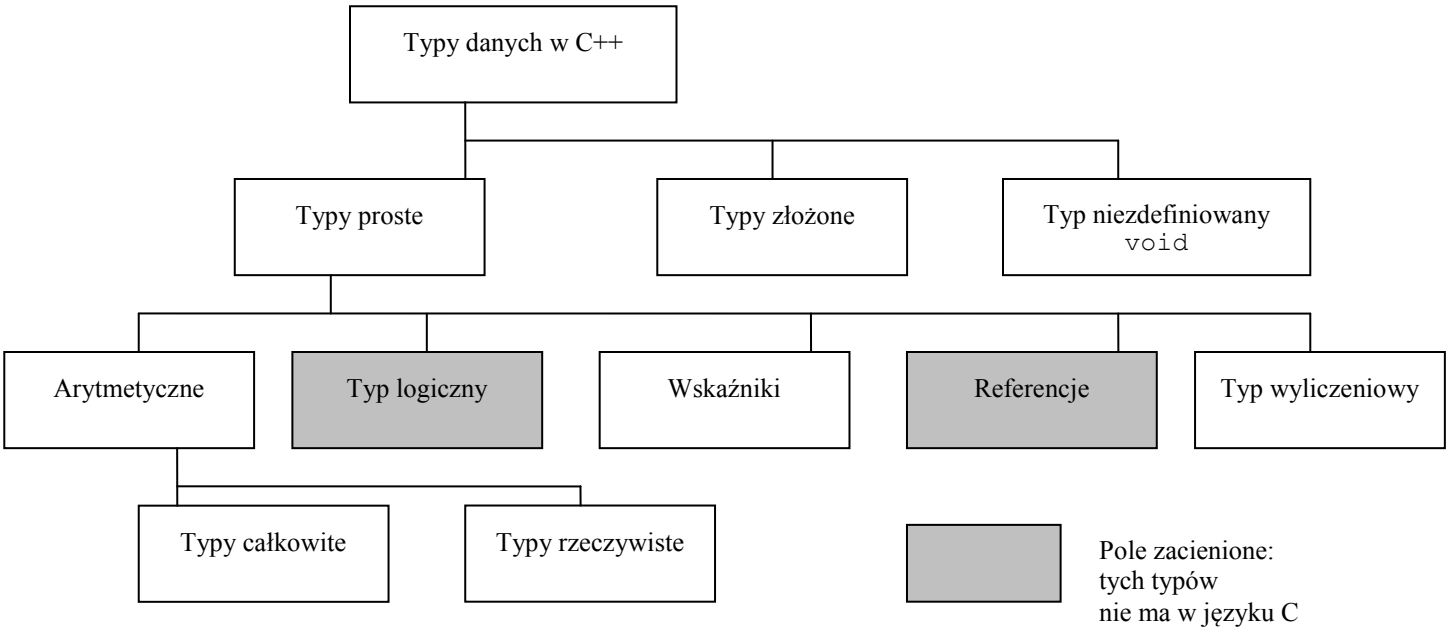
- Ustalając nazwy dobrze jest stosować kilka ogólnie przyjętych zasad:
  - powinno to być słowo kojarzące się z przeznaczeniem zmiennej,
  - najczęściej używa się małych liter, np. `indeks` nie zaś `INDEKS`,
  - jeśli nazwa składa się z kilku słów, poszczególne słowa oddziela się znakiem podkreślenia, albo zaczyna się każde słowo od wielkiej litery oprócz pierwszego,
  - wybrany styl tworzenia nazw powinien być jednorodny w całym programie.

2.3. Słowa kluczowe

- W języku C++ słowa o zastrzeżonym znaczeniu nazywane są *słowami kluczowymi* (ang. *keywords*). Jest ich 63. Tych słów nie można używać jako nazwy zmiennych i innych obiektów.
- Przykłady słów kluczowych:
  - `int` - używane jest do określenia typu całkowitego
  - `if` - instrukcja wyboru
  - `sizeof` – operator zwracający rozmiar argumentu w bajtach

2.4. Podstawowe typy danych

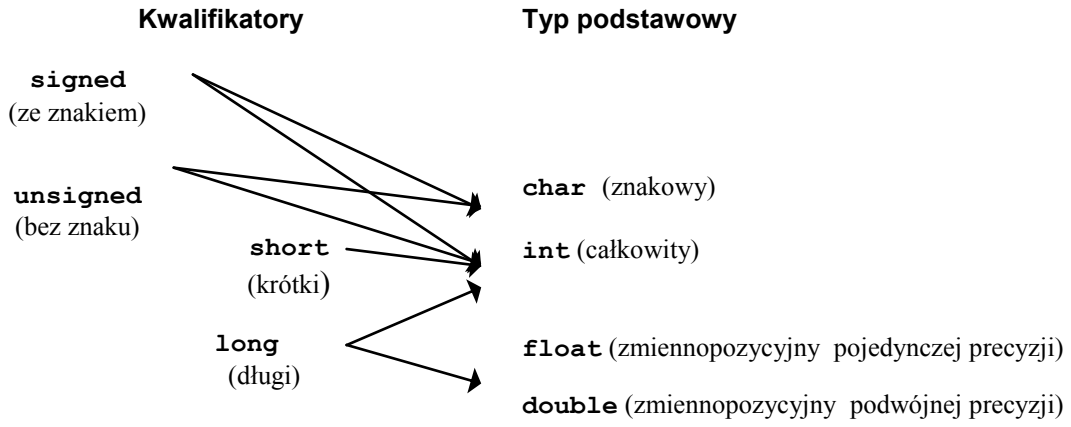
- Dane (zmienne, stałe, wartości wyrażeń, wartości generowane przez funkcje) przechowywane są w pamięci jako ciąg bitów.
- *Typ danych* - nadaje znaczenie ciągowi bitów o określonym adresie i długości:
  - określa ile pamięci potrzeba do przechowania danej,
  - jakie operacje mogą być wykonane na danej,
  - jak te operacje są interpretowane.



- Zbiór typów prostych dostępnych w języku C++ i C jest podobny.
- Typy *proste* (skalarne, ang. *scalar types*) są niepodzielne, wykorzystuje się je do przechowywania pojedynczych danych.
  - *typy arytmetyczne*: dla liczb całkowitych, rzeczywistych, znaków,
  - *typ logiczny*: dla wartości logicznych {*prawda*, *fałsz*} (nie ma go w C)
  - *typ wskaźnikowy*: dla adresów obiektów danego typu
  - *typ referencyjny*: dla innej nazwy obiektu (nie ma go w C)
  - *typ wyliczeniowy*: dla reprezentowania zbioru wartości podanych przez użytkownika.
- Typy *złożone* (ang. *aggregate types*): składają się z elementów typów prostych oraz innych typów złożonych. Wykorzystywane do przechowywania danych powiązanych ze sobą. Do typów złożonych należą:
  - *tablice* obiektów danego typu,
  - *struktury* (i *klasy*) zawierające zestawy obiektów różnego typu,
  - *unie* zawierające dowolny z zestawu obiektów o różnych typach,
  - *funkcje* zwracające wartości danego typu.
- Typ `void`: typ ten ma specjalne zastosowania, nie ma obiektów typu `void`; używany jest na przykład wtedy, kiedy chcemy powiedzieć, że funkcja nie zwraca żadnej wartości.

2.4.1. Typy arytmetyczne

- Typy arytmetyczne służą do przechowywania liczb całkowitych, rzeczywistych i znaków.
- Są one dostępne w różnych rozmiarach, dzięki czemu programista może wybrać ilość zużywanej pamięci, precyzję i zakres przechowywanych liczb.



- Rozmiary i zakresy danych poszczególnych typów mogą być różne dla różnych implementacji kompilatora.
- Dla typów `int` i `char` określana jest najmniejsza i największa wartość liczby, która może być przechowywana w zmiennej tego typu.
- Dla typów `float` i `double` określane są:
  - największa liczba możliwa do przedstawienia
  - najmniejsza liczba możliwa do przedstawienia
  - rozróżnialność - najmniejsza dodatnia liczba x taka, że $1.0+x \neq 1.0$
  - dokładność - liczba cyfr dziesiętnych, które mogą być przedstawione dokładnie
- Największe i najmniejsze wartości każdego typu w danej implementacji są podane w standardowych plikach nagłówkowych: `climits` i `cfloat`.
- (Uwaga: w nowszych implementacjach ograniczenia są dostępne w klasie wzorcowej `numeric_limits<>`).

- Niektóre nazwy typów można podawać w postaci pełnej i skróconej. Na przykład mówiąc o typie `signed int` najczęściej pomija się kwalifikator i stosuje się nazwę krótszą `int`.

Długi format nazwy typu	Krótki format nazwy typu
`char`	`char`
`signed char`	`signed char`
`unsigned char`	`unsigned char`
`signed short int`	**`short`**
`unsigned short int`	**`unsigned short`**
`signed int`	**`int`** (lub **`signed`**)
`unsigned int`	**`unsigned`**
`signed long int`	**`long`**
`unsigned long int`	**`unsigned long`**
`float`	`float`
`double`	`double`
`long double`	`long double`

- Standard C++ ANSI/ISO określa tylko minimum jakie musi spełniać dany typ:

Nazwa typu	Zakresy przechowywanych wartości (standard ANSI/ISO)	Do czego służy
Typy całkowite		
`char`	Musi być wystarczająco duży, aby można było przechować w nim dowolny znak ze zbioru wspieranego przez implementację, np. 127 znaków zbioru podstawowego ASCII lub 255 znaków zbioru rozszerzonego ASCII. Zmienna typu char przyjmuje wartości takie jak `signed char` lub `unsigned char` w zależności od implementacji.	bardzo małe liczby całkowite, kody znaków, np. ASCII
`signed char`	-127 do 127	bardzo małe liczby całkowite, kody znaków, np. ASCII
`unsigned char`	0 do 255	bardzo małe liczby całkowite dodatnie, kody ASCII
`int`	Obiekty typu `int` zajmują jedno słowo. Jeśli w danym systemie operacyjnym przyjęte jest słowo 16 bitowe oznacza to liczby z zakresu --32 768 do 32 767; Jeśli zaś słowo wynosi 32 bity, zakres liczb wynosi od -2 147 483 648 do 2 147 483 647	średnie lub duże liczby całkowite
`short int`	Co najmniej połowę słowa; jednak w komputerach 16 bitowych przyjęte jest, że typ `short` i `int` mają ten sam rozmiar	średnie liczby całkowite
`long int`	Co najmniej tyle co `int`. W komputerach 16 bitowych przyjęte jest używać dwa słowa, w komputerach 32 bitowych zazwyczaj `int` i `long int` mają te same rozmiary	duże liczby całkowite
`unsigned int`	Taki sam rozmiar jak dla `int`. Dla `int` 16 bitowego oznacza to wartości od 0 do 65 535 ; zaś dla `int` 32 bitowego od 0 do 4 294 967 295	średnie lub większe liczby całkowite dodatnie
`unsigned short int`	Taki sam rozmiar co `short int`; Dla `short` 16 bitowego oznacza to wartości od 0 do 65 535.	średnie liczby całkowite dodatnie
`unsigned long int`	Taki sam rozmiar co `long int`; Dla `long int` 32 bitowego oznacza to wartości od 0 do 4 294 967 295	bardzo duże liczby całkowite dodatnie
Typy rzeczywiste		
`float`	zależy od implementacji, zbiór wartości `float` stanowi podzbiór wartości `double`	liczby rzeczywiste przedstawione z pojedynczą dokładnością – przykładowo małe liczby rzeczywiste o 7 cyfrach dokładności
`double`	zawiera co najmniej zbiór wartości `float`, co najmniej taka dokładność jak `float`,	liczby rzeczywiste przedstawione z podwójną dokładnością – przykładowo duże liczby rzeczywiste o 15 cyfrach dokładności
`long double`	zawiera zbiór wartości `double`, co najmniej taka dokładność jak `double`,	bardzo duże liczby rzeczywiste (18 cyfr dokładności)

- Przykład: program wyświetla rozmiar typów (w bajtach) w danym kompilatorze:

```
#include <iostream>
using namespace std;
int main()
{
  cout << endl;
  cout << "ROZMIAR TYPU " << endl;
  cout << "------------ " << endl;
  cout << "int          " << sizeof(int) << endl;
  cout << "long         " << sizeof(long) << endl;
  cout << "float        " << sizeof(float) << endl;
  cout << "double       " << sizeof(double) << endl;
  return 0;
}
```

2.4.2. Typ logiczny bool

- Typ logiczny służy do przechowywania wartości logicznych.
- Zmienne tego typu mogą przyjmować dwie wartości: `true` (prawda) i `false` (fałsz).
- Typ logiczny wprowadzono w języku C++. W języku C do wyrażenia wartości logicznych wykorzystywany jest typ `int`. Przyjęto następującą konwencję: 0 oznacza fałsz, wartość różna od zera oznacza prawdę.

2.5. Nadawanie typu zmiennym (deklarowanie) i stałym

- ***Typ i nazwę zmiennej (ang. variable) określa użytkownik jawnie, za pomocą odpowiedniej deklaracji, biorąc pod uwagę dane, które mają być w zmiennej przechowywane.***
- Deklaracja typu składa się ze specyfikatora typu i występującej po niej nazwy zmiennej. Musi być zakończona średnikiem. Jeśli definiujemy kilka zmiennych tego samego typu można je umieścić w jednej deklaracji typu, oddzielając nazwy przecinkami.
- Przykłady:

```
int licznik;
int a,b,c;
double suma;
char znak;
bool flaga;
```

- ***Deklaracja typu musi poprzedzać w tekście programu użycie zmiennej.***
- W języku C deklaracja typu może występować tylko na początku funkcji (programu lub bloku), przed instrukcjami związanymi z wykonywaniem programu.
- W języku C++ deklaracja typu może występować w dowolnym miejscu tekstu programu, byle przed użyciem zmiennej.

- ***Typ stałej (ang. literal constant) określa użytkownik odpowiednio zapisując daną w instrukcji.***

```
// Wykorzystanie stałych do zainicjowania zmiennych
  int licznik = 0;    // stała 0 typu int: liczba bez kropki dziesiętnej
  double suma = 0.;   // stała 0 typu double: liczba z kropką dziesiętną
  char c='*';         // stała * typu char: znak z apostrofach
  bool b=false;       // stała false typu bool: jedna z dwóch wartości false
                                                        // lub true
```

2.6. Nadawanie wartości zmiennym

- Zmienna przyjmuje wartości w wyniku *inicjowania*, *przypisania* lub *wczytania*.

- ***Inicjowanie***: jest to nadanie pierwszej wartości zmiennej podczas jej definiowania, nie jest obowiązkowe:

```
int licznik = 0;
int ile,liczba=20;  /* ile nie ma wartości początkowej */
float powierzchnia=4.5;
double suma = 0.;
long odl_od_Ksiezyca=238857;
char c='*';
bool b=false;
```

- Zmienna, której nie nadano wartości początkowej to zmienna *niezainicjowana* (ang. *uninitialized*).
- Zmienna *zdefiniowana niezainicjowana* ma pewną wartość, ale wartość ta jest nieokreślona - wynik tego, że podczas rezerwowania obszaru pamięci dla zmiennej, pamięć ta nie jest czyszczona (*Uwaga*: są wyjątki - pewne zmienne są automatycznie inicjowane z wartością 0, patrz wykład na temat modeli pamięci).

```
// Przykład błędnie działającego programu
#include <iostream>
int main()
{
  int k;
  cout << "Wartosc k: " << k << endl;
  return 0;
}
```

Program się skompilował i można było go uruchomić. Niektóry kompilator mógł wysłać ostrzeżenie:: `Possible use of 'k' before definition.` Jednakże wynik wykonania programu jest przypadkowy:

```
Wartosc k: 1134518940
```

- ***Przypisanie***: w trakcie działania programu zmiennej przypisuje się wartość za pomocą operatora przypisania = :

```
int ile;
float powierzchnia;
ile=20;               // przypisanie
powierzchnia=4.5;     // przypisanie
```

- ***Wczytanie***: w trakcie działania programu można wczytywać wartości do zmiennych na przykład za pomocą konstrukcji `cin >> nazwa_zmiennej`:

```
int ile;
cin >> ile;
```

2.7. Stałe

- *Stałe* (*literały*, ang. *literal constant*) reprezentują ustalone wartości, które nie mogą być zmienione podczas działania programu.
- Z każdą stałą związany jest jej typ i wartość.

2.7.1. Stałe liczbowe - całkowite i zmiennopozycyjne

- *Stała całkowita* to liczba całkowita, ewentualnie poprzedzona znakiem.
- Domyślnym typem *stałych całkowitych* jest typ `signed int` . Przykłady:
  `0  1234  -1234`
- *Stała zmiennopozycyjna* to liczba zawierająca kropkę dziesiętną (w przypadku notacji wykładniczej kropka może być pominięta), ewentualnie wykładnik, może być poprzedzona znakiem.
- Domyślnym typem *stałych zmiennopozycyjnych* jest typ `double`. Przykłady:
  `3.14  3.  0.14  .14  1.2e5  2e-5  0.0  0.`
  gdzie:
  `1.2e5` oznacza $1{,}2*10^5$
  `2e-5` oznacza $2*10^{-5}$
- Pozostałe typy wymagają wpisania odpowiedniego specyfikatora, zgodnie z poniższym zestawieniem:

Stała	Typ stałej
`1000L`	`long int` - litera `L` za liczbą całkowitą
`1024l`	`long int` - litera `l` za liczbą całkowitą
`128u`	`unsigned int` - litera `u` za liczbą całkowitą (mała lub wielka)
`1010LU`	`unsigned long int` - litery `LU` za liczbą (w dowolnej kolejności)
`3.14f`	`float` - litera `f` za liczbą z kropką dziesiętną (mała lub wielka)
`1.0L`	`long double` - litera `L` za liczbą z kropką dziesiętną (mała lub wielka)

Różne zapisy stałej całkowitej

- Stałe całkowite można zapisywać na trzy sposoby:
  - dziesiętnie: `20`
  - ósemkowo: `024` - liczba jest poprzedzona cyfrą 0 (dziesiętnie ma wartość 20)
  - szesnastkowo: `0x14` - liczba jest poprzedzona znakami 0x (lub 0X)
- Dotyczy to *wszystkich* typów stałych całkowitych. Przykład: `0x12345LU`

2.7.2. Stałe logiczne

- Stałe logiczne mają dwie wartości: `true` i `false`.

2.7.3. Stałe znakowe

- Stała znakowa (ang. *character constant*) to znak umieszczony w *apostrofach*: `'a'`
- Typem stałej znakowej jest `char`.
- Wartość stałej znakowej to *kod liczbowy* znaku. Do reprezentacji znaków używany jest kod ASCII.
  Przykłady:
  'a' (kod 97) - kompilator zapamiętuje znak `'a'` jako liczbę 97
  'A' (kod 65)
  `'0'` (kod 48)'
  `'8'` (kod 56)
  `' '` (kod 32)
- Znak można podawać również w postaci liczby całkowitej - kodu ASCII znaku:

```
char znak;      // deklaracja zmiennej typu char
znak='a';       // przypisanie wartości kodu znaku 'a'
znak=97;        // to samo - kod podany dziesiętnie
```

- Znaki niedrukowalne wymagają poprzedzenia znakiem odwróconego ukośnika (ang. *backslash*).
  Przykłady:
  `'\n'` - koniec linii (ang. *new line*), kod ASCII 10
  `'\r'` - przesunięcie do początku wiersza (ang. *carriage return*), kod ASCII 13
  `'\a'` - sygnał dźwiękowy (ang. *alert*), kod ASCII 7
  `'\0'` - znak pusty (ang. *null*), kod ASCII 0
- Znak ukośnika można użyć wraz z ósemkowym lub szesnastkowym przedstawieniem kodu znaku.
  Przykład: różne sposoby przypisania znaku `'a'` do zmiennej `znak`

```
char znak;
znak='a';       // przypisanie wartości kodu znaku 'a'
znak=97;        // to samo - kod podany dziesiętnie
znak=0141;      // to samo - kod podany ósemkowo
znak=0x61;      // to samo - kod podany szesnastkowo
znak='\141';    // to samo - kod ósemkowy
znak='\x61';    // to samo - kod szesnastkowy
```

- Jeśli chcemy przedstawić ukośnik to musimy użyć sekwencji `'\\'` .

2.7.4. Stałe napisowe

- Stała napisowa (*łańcuch*, *literał*, *tekst* ang. *string constant*, *string literal*) jest to ciąg znaków umieszczony w cudzysłowach. Przykład:

```
"Witamy"
```

- Typem stałej znakowej jest *tablica* typu `char` (ciąg bajtów przylegających do siebie).
- Stała napisowa jest pamiętana jako sekwencja znaków zakończona znakiem pustym (`'\0'`).
- Więcej na temat stałych znakowych podczas omawiania tablic i wskaźników.
- Przykłady wykorzystania stałych napisowych:

```
cout << "Witamy!";
cout << "Witamy!\n";
cout << "Witamy\nw szkole\n");
cout << "Lp.\tNazwisko";
```

2.8. Modyfikator typu `const`

- Modyfikator `const` przekształca obiekt w *stałą* (ang. *constant*). Wartość takiego obiektu określana jest podczas definiowania i nie może później ulec zmianie. Jest przeznaczona tylko do czytania (ang. *read only*).
- Często tego typu obiekt nazwa się stałą symboliczną (ang. *named constant*), w odróżnieniu od stałej zapisywanej dosłownie (ang. *literal constant*) np. w postaci liczby.
- Przykład:

```
const int id=12345;
const int max=100;
...
id=10000;  // BŁĄD! Próba przypisania nowej wartości
int tab[max]; // Poprawne użycie: stała może być
              // używana do określenia wymiaru tablicy
```

- Wartość stałej może być określana za pomocą innej stałej.

```
const int odlegloscMile = 3959;
const int odlegloscKm = 1.609*odlogloscMile;
```

- Zaletą stosowania stałych symbolicznych jest łatwiejsza pielęgnacja kodu programu. W razie zmiany wartości stałej wystarczy zmienić wiersz z jej definicją.
- Eliminujemy w ten sposób istnienie w programie magicznych liczb, reprezentujących jakieś założenie odnośnie programu, np. jakieś przeliczniki lub rozmiary tablic. Jeśli założenia te będziemy zapisywać za pomocą dobrze skomentowanych stałych symbolicznych, łatwiej będzie zmieniać czy weryfikować program.
- Z informacji o tym, że pewna wartość nie zmienia się podczas wykonywania programu może również korzystać kompilator budując bardziej efektywny kod.

2.9. Definiowanie nazw typów - `typedef`

- Za pomocą konstrukcji `typedef` można nadać nową nazwę (synonim) istniejącemu typowi. Jeśli na przykład wpiszemy następującą definicję:

  ```
  typedef float realType
  ```

  Oznacza to, że można teraz wymiennie używać `float` i `realType`:

  ```
  float x;
  realType y;
  ```

Zastosowania

- Wprowadzenie własnej nazwy czyni program łatwiejszym do modyfikacji. Załóżmy, że w programie działania na liczbach rzeczywistych zaprogramowano z użyciem typu `float`. Podczas użytkowania programu stwierdzono, że wymagana jest większa precyzja i potrzebny jest typ `double`. Oznacza to, że trzeba wszystkie wystąpienia zmiennych typu `float` należy zamienić na `double`. Zmiany te byłyby prostsze do wykonania, jeśli użyto by instrukcji `typedef`. Na przykład:

  ```
  #include <iostream>
  int main()
  {
    typedef float real; // real jest synonimem float

    const real PI=3.14159;
    real promien;
    cout << "Wpisz promien:";
    cin >> promien;
    real obwod=2*PI*promien;
    cout << "Obwod=" << obwod << endl;
    return 0;
  }
  ```

  Zmiana typu zmiennych używanych do obliczeń sprowadzi się do wymiany instrukcji

  ```
  typedef float real
  ```

  na:

  ```
  typedef double real;
  ```

- Nowa nazwa może być wygodnym skrótem dla określenia typu o długiej nazwie:

  ```
  typedef unsigned char uchar;
  ```

  Odtąd zamiast `unsigned char` można pisać `uchar`.

- Standardowe biblioteki C++ bardzo często korzystają z tej techniki w przypadku deklaracji, które zależą od systemu. Łatwiej jest wtedy przenosić oprogramowania między różnymi komputerami.

  Przykład:
  Załóżmy, że w pewnych zastosowaniach sieciowych potrzebujemy posługiwać się liczbami umieszczonymi na 32 bitach (przykładem jest adres IP). Moglibyśmy skorzystać z typu `unsigned int`, ale typ `int` może zajmować 2 (czyli 16 bitów) lub 4 bajty (32 bity) w zależności od komputera. Chcemy napisać program, który łatwo będzie można przenosić. Załóżmy, że nasz komputer przeznacza na typ `unsigned int` 32 bity. Możemy zatem zdefiniować typ o nazwie `uint32` jako:

  ```
  typedef unsigned int uint32
  ```

  Tym typem będziemy się posługiwać wszędzie tam, gdzie będziemy definiowali zmienne wymagające 32 bitów. Jeśli teraz program zostanie przeniesiony na komputer, na którym `unsigned int` zajmuje 2 bajty, zaś `unsigned long int` zajmuje 4 bajty, wystarczy w odpowiednim pliku nagłówkowym zmienić jedną definicję:

  ```
  typedef unsigned long int uint32
  ```

2.10. Przykład

Poniższy program ilustruje co się dzieje jeśli użyjemy liczby spoza zakresu (zbyt małej lub zbyt dużej). Skomentuj wyniki uzyskane w programie.

```
#include <iostream>
#include <climits>
#include <cfloat>
using namespace std;
int main()
{
    int zm_int;

    zm_int = INT_MIN;
    cout<<"Minimalna liczba calkowita typu int       :"<<zm_int<<endl;
    zm_int = INT_MIN - 25;
    cout<<"Minimalna liczba calkowita typu int - 25  :" << zm_int << endl;
    zm_int = INT_MAX;
    cout<<"Maksymalna liczba calkowita typu int      :" << zm_int << endl;
    zm_int = INT_MAX + 25;
    cout<<"Maksymalna liczba calkowita typu int + 25 :" << zm_int << endl;

    cout<<"Niedomiar (underflow) zmiennoprzecinkowy  :"
                                          << (double) DBL_MIN*0.1 << endl;
    cout<<"Nadmiar (overflow) zmiennoprzecinkowy    : "
                                          << (double) DBL_MAX*10 << endl;

}
```